BEA WebLogic Server

Using the Zero Administration Client (ZAC) (Deprecated)

WebLogic Server Version 6.0
Document Edition 1.0
March 2001

Using the Zero Administration Client

Document Edition	Date	Software Version
6.0	March 2001	BEA WebLogic Server 6.0

Contents

3. Developing with WebLogic ZAC

CHAPTER

1 Publishing with WebLogic ZAC

This section provides an overview of using the WebLogic Zero Administration Client (ZAC), including the following topics:

- Introduction
- Trying Out the ZAC Demos
- How ZAC Works
- Setting up WebLogic for Publishing with ZAC

Introduction

Note: The WebLogic Zero Administration Client is a deprecated product. BEA recommends that you use the Sun Microsystems Java Web Start product. Java Web Start is a Java 2-compliant product that enables users to download Java applications.

WebLogic ZAC, the Zero Administration Client utility, lets you publish and republish applications, applets, and libraries with ZAC, so that they are transparently and automatically updated to the latest version on the end user client machine. With ZAC, you no longer need to manually distribute applications, applets, or libraries to your clients; you can depend on ZAC's automatic services to do so.

ZAC is extremely efficient. When a ZAC application is republished, only the minimal amount of data is sent over the network to each client, to bring the applications on your clients up-to-date. In a typical scenario where little or nothing has changed, the overhead for checking for new files at startup is not noticeable to a user.

ZAC is highly configurable, so that you can design how your application should be published, installed, and updated. You can check for updates to the application and to its dependent libraries each time the application starts or stops; on a scheduled basis, or you can disable the check for new ZAC updates altogether. Although this would disable ZAC's most powerful feature, it may be desirable for packages that you intend to distribute once only as a static version.

ZAC uses a protocol called the HTTP Distribution and Replication Protocol (DRP), a specification submitted to the W3C in August 1997 for the efficient replication of data over HTTP.

This document includes instructions on how to use the ZAC Publish Wizard to publish applications on a WebLogic Server for distribution to your users.

Trying Out the ZAC Demos

When you **install on Windows with the install shield (.exe) version**, WebLogic comes out-of-the-box with two ZAC packages, ZSimple and ZUpdate, that you can try out immediately to see how WebLogic ZAC works. (This pre-installed demo does not function correctly if you have installed WebLogic Server from the .zip file distribution.)

1. Start WebLogic. (You will need to know the system password to publish.)
2. Start the ZAC Publish Wizard. Win32 users can use the shortcut for *ZAC Publisher* available in the WebLogic directory in the Start menu.

Non-Windows users can start the Publish Wizard from the command line (after setting your CLASSPATH with this command:

```
$ java weblogic.PublishWizard
```

3. From the Publish Wizard window, double-click the ZSimple or ZUpdate selection. You can walk through all the steps to see a demonstration of how to publish a package, or you can just press the *Finish* button to skip to the review phase.

4. Press the “Done” button.

5. To publish the application, select it in the list and select *Publish...* from the Package menu. Select the appropriate Host and press the *Publish* button. If you are only connected to one host, this will be selected automatically for you. A “Publishing...” dialog will display the status of the publish operation and, if successful, the package will appear in the lower-left window of the Publish Wizard, under the selected WebLogic Server host. You may need to expand the “+” symbol to see the published package.

6. You can test the published package by selecting the package under a WebLogic Server host in the published window (lower-left), then select *Test run* from the Server menu.

You can open the package again after the test run starts, change a few parameters, and republish the application to see how the client responds.

The rest of this document describes how to create and publish a package, and how to create a bootstrap executable — which is the standard way to install and run an application from a ZAC package.

How ZAC Works

Any Java application, applet, or library can be published as a ZAC package. You do not need to add anything special to the Java source code to publish your program with ZAC. (However, ZAC does include a Java API that you can use in writing your application to add interactive control over when ZAC updates should occur or whether the application should respond immediately to ZAC updates. Developing with the ZAC API is discussed in Developing with WebLogic ZAC.)

ZAC works very simply from your user’s perspective.

1. You publish an application to a WebLogic Server with the ZAC Publish Wizard.

2. Your user downloads a small native-OS installer, and simply double-clicks to install and start.
3. The installer creates your application and installs all necessary libraries to support it on the user's machine, as well as a small bootstrap that monitors for new published versions of your application and carries out updates.

How you Publish a ZAC Package on the Server

The ZAC Publish Wizard makes publishing your application easy. You use the ZAC Publish Wizard to:

- Create new ZAC packages
- Publish ZAC packages on a WebLogic Server
- Import ZAC packages from other WebLogic Servers
- Update published ZAC packages on a WebLogic Server

The ZAC Publish Wizard guides you through the process of creating and publishing a package with ZAC on a WebLogic Server. During the publish process, you set up the parameters necessary to run your application or ZAC package on the client machine, such as identifying:

- Which directories contain the files that comprise your application
- Other ZAC packages that your application depends upon
- Which Java runtime environment must be present on the client machine
- Other necessary information to allow your application to work correctly

An application is published to a WebLogic Server and is made available to your users via HTTP. You can republish the package each time you change your application or any of the libraries on which it depends.

How ZAC Installs a Published Application on the User's Machine

When you complete the publishing phase, your application is published on the WebLogic Server. The ZAC Publish Wizard generates a small installation program in native format for each machine type supported by ZAC. Your users download and run

this program to install the published application or package. We shall refer to this installation program as the 'installer', in the rest of this document. The installer is a very small executable, and so is quick to download.

To make the published application available to users, just attach the installer to an email or embed an FTP link to it in an HTML page. The user downloads and runs the installer. Since the installation program is a native executable, a user doesn't need to pre-install a Java Runtime Environment (either a JRE or some Java development environment like the JDK).

The installer performs some or all of the following tasks, depending on how you have configured it from the ZAC Publish Wizard:

- (*Optional*) Checks that a specific JRE that will be needed by the published application is pre-installed on the client machine; the ZAC installer can automatically install a JRE that you provide (as another ZAC package) if necessary.
- (*Optional*) Queries the user for HTTP proxy information to connect to the publishing WebLogic Server.
- (*Always*) Downloads and installs the ZAC application.
- (*Optional*) Downloads and installs any other ZAC packages that the published application is dependent upon.
- (*Always*) Installs a bootstrapper executable that is used to start a published application, to monitor the publishing WebLogic Server for new versions, and to download new versions when appropriate.
- (*Optional*) Creates a desktop icon or Start menu item that links to the bootstrapper.
- (*Optional*) Launches the ZAC application.

After the initial installation, the installer may be deleted from the client's machine.

How a Published Application Runs on the User's Machine

The installer installs your user application, as well as a native bootstrap file. Once the application is installed, your user will use the native-OS bootstrap to invoke the application. Part of the ZAC Publish Wizard process involves creating the bootstrap program.

Each time the client runs the bootstrapper, it first checks for new versions of the published application on the WebLogic Server. If a new version of your application — or any other ZAC packages that the ZAC application depends upon — has been published, the bootstrapper automatically updates the ZAC packages on the client machine with the new versions. The bootstrapper then starts the published application.

Setting up WebLogic for Publishing with ZAC

ZAC must be deployed as a Web application on the WebLogic Server for users to have access to it. Servlets included within the ZAC Web application handle requests from clients. ZAC is deployed on the server as a WAR file, `zac.war`. To prepare the ZAC package for deployment, do the following:

1. Edit the file `web.xml` (part of the ZAC Web application) to set a publish root for locating published packages. If the publish root is set to a relative path, the directory will be located in the WebLogic home directory (parallel to the `myserver\` directory). If unset, the publish root defaults to the `exports\` directory in WebLogic home). The following is an example of an entry in web.xml that sets the publish root:

   ```
   <context-param>
    <param-name>weblogic.zac.publishRoot</param-name>
    <param-value>C:/weblogic/publish</param-value>
   </context-param>
   ```

2. Optionally, set an ACL for each published package to limit access. If unset, the write permission defaults to system, and the read permission defaults to everyone. Setting an Access Control List is also accomplished by editing the `web.xml` file that is included with the `zac.war` application. The following is an example of an entry in the `web.xml` file that sets an ACL for `myApp` that limits access to three users, Peter, Paul and Mary:

   ```
   <context-param>
   <param-name>weblogic.allow.read.weblogic.zac.myApp</param-name>
    <param-value>Peter,Paul,Mary</param-value>
   </context-param>
   ```

To deploy the ZAC Web application, do the following:

1. Start the WebLogic Server as usual. The WebLogic Server must be running when you can actually publish your ZAC packages, though it need not be running when you are creating the ZAC packages.
2. Invoke the WebLogic Administration Console.
3. To make published packages available, deploy the ZAC Web application archive file `zac.war` as a Web application. For information on deploying Web applications, see Assembling and Configuring Web Applications in the WebLogic Server Administration Guide.

CHAPTER

2 Using the Publish Wizard

This section discusses how to use the Publish Wizard, including the following topics:

- Starting the Publish Wizard
- Creating a ZAC Package
- Publishing a ZAC Package
- Using the command line Publish Utility
- Connecting the Publish Wizard to Other Servers
- Updating a Published ZAC Package
- Importing a Published ZAC Package from Another Server
- Removing a Published ZAC Package
- Creating an Installer/Bootstrap Application
- Packaging a JRE

Starting the Publish Wizard

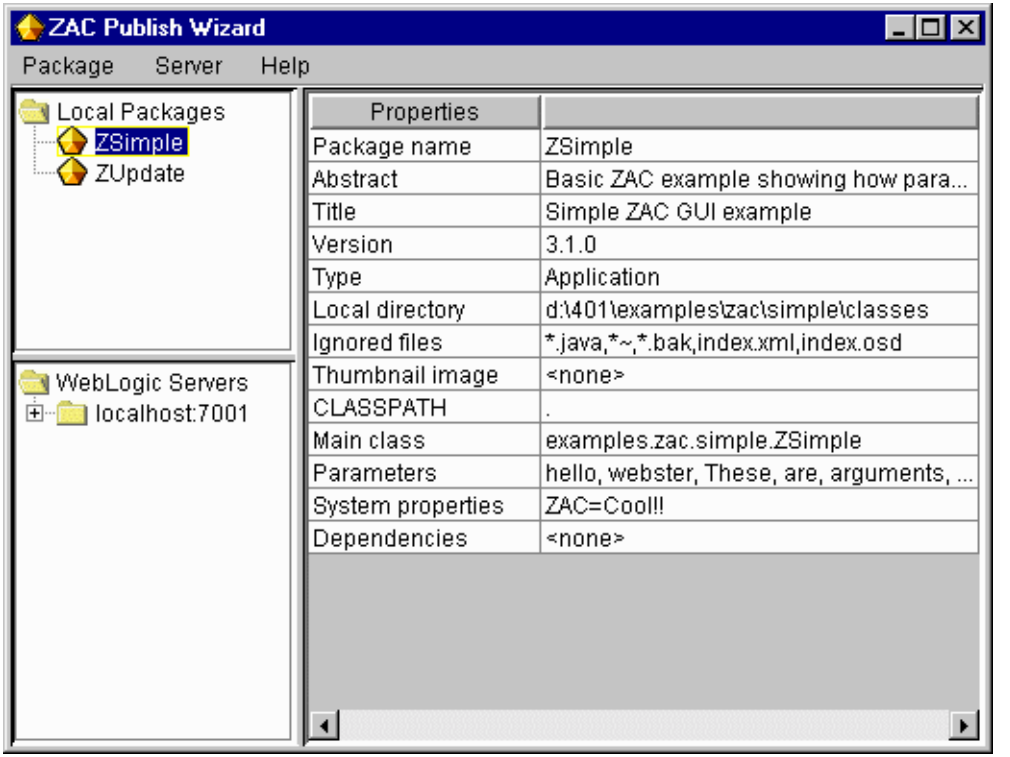
If you wish to publish to a WebLogic Server, that server must be running, you must have configured a DefaultWebApp for the server, and you will need to know a user and password that has permission to publish. For more details, see Publishing a ZAC package. However, you can create and inspect existing ZAC packages without any Servers running.

Users can start the Publish Wizard from the command line (after setting your CLASSPATH with this command:

```
$ java weblogic.drp.admin.PublishWizard
```

The ZAC Publish Wizard splash screen will appear while ZAC starts up, then the main dialog for the Publish Wizard will appear:

Figure 2-1 The ZAC Publish Wizard Main Window



Creating a ZAC Package

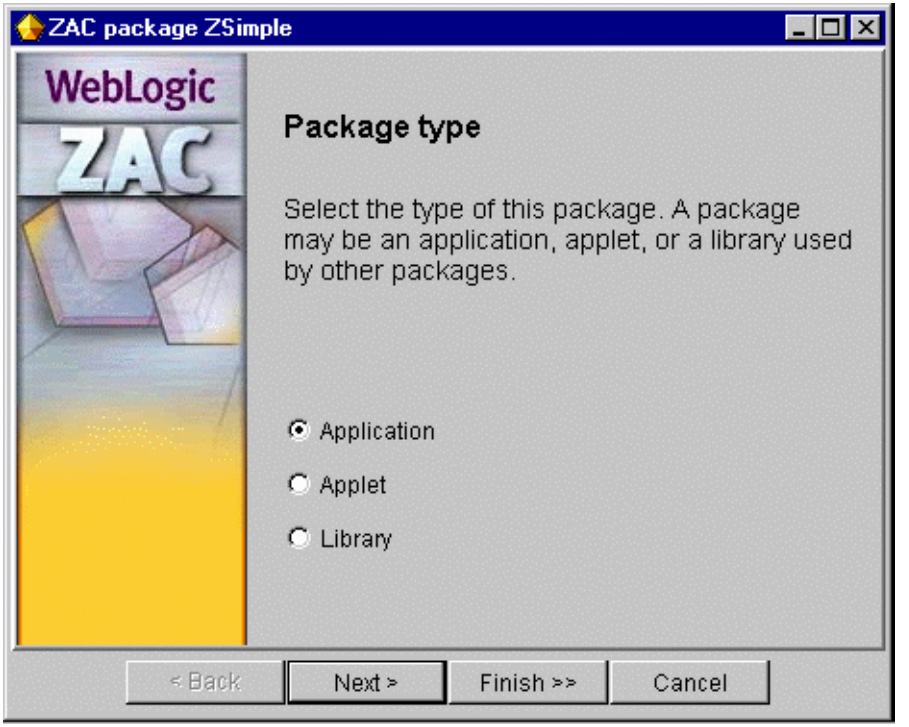
You use the ZAC Publish Wizard to create new ZAC packages, import existing packages from other servers, or update previously published packages. A ZAC package may be an application, applet, or library.

The Publish Wizard will guide you through the process of creating a new package or republishing an existing. You can also use ZAC's "test run" option, which allows you to try running your ZAC application to check that it will run as configured.

1. From the Package menu, select *New* to create a new package, or *Open* to view or edit an existing package. Alternatively, you can double-click on an existing package. This will open the Package panel.

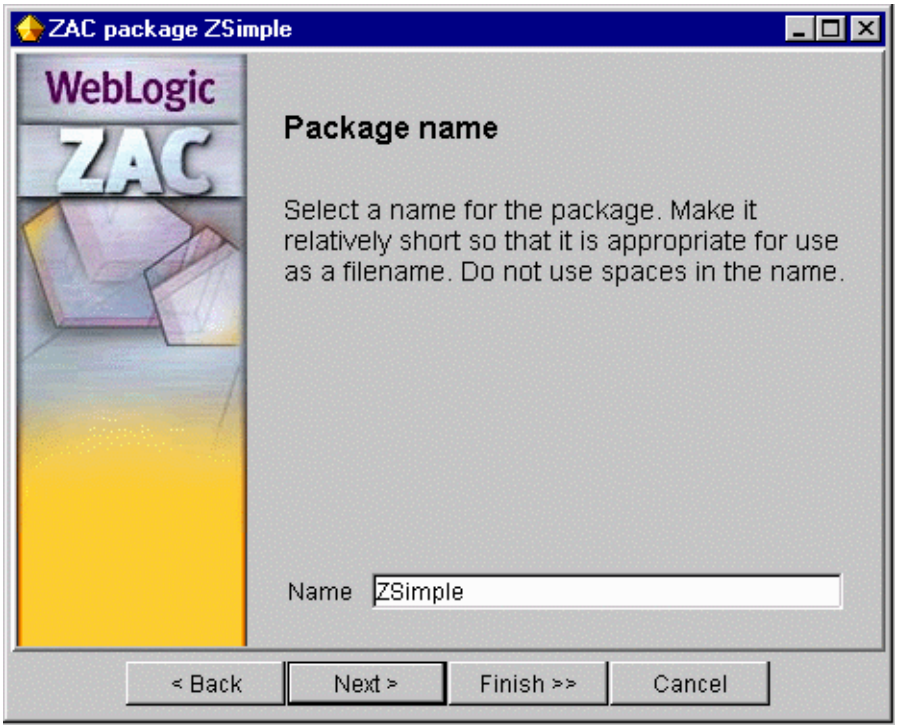
2. On the Package panel, select a package type. You may publish an applet, a Java application, or a class library as a ZAC package. Typically, you publish libraries as a component that other published applications depend upon.

Figure 2-2 Publishing a New Library Package



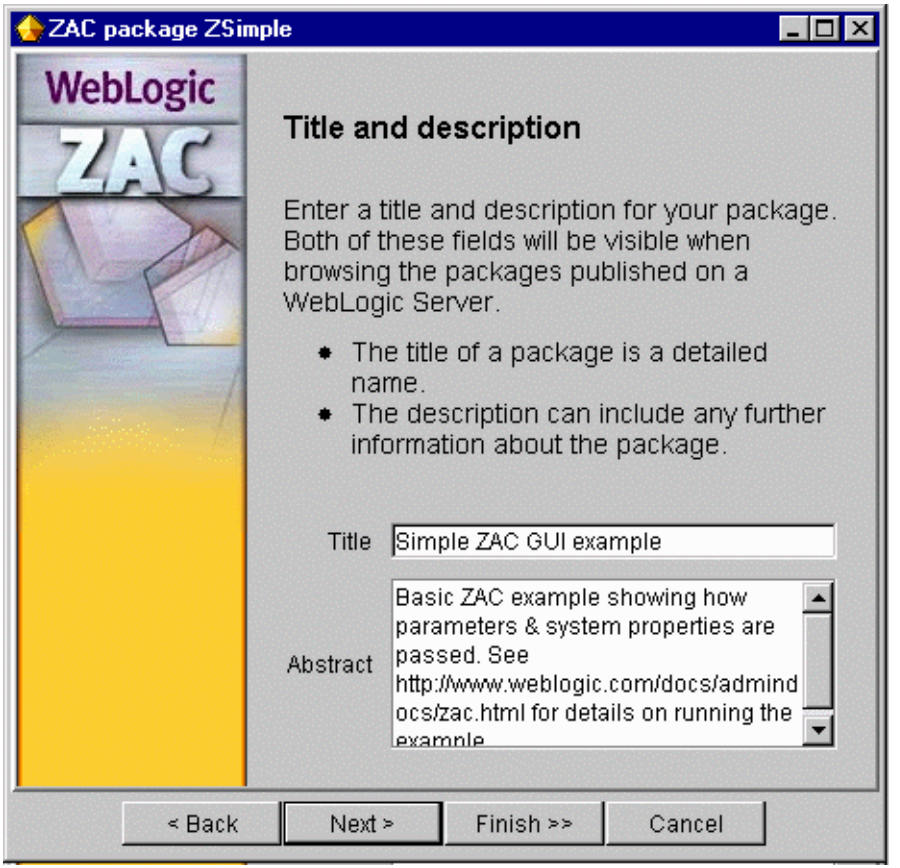
3. Name the package. This will not be a public display name; it will be used for filename purposes, so make it short and easy to identify. **Note**: You may not use spaces in the package name.

Figure 2-3 Naming the Package



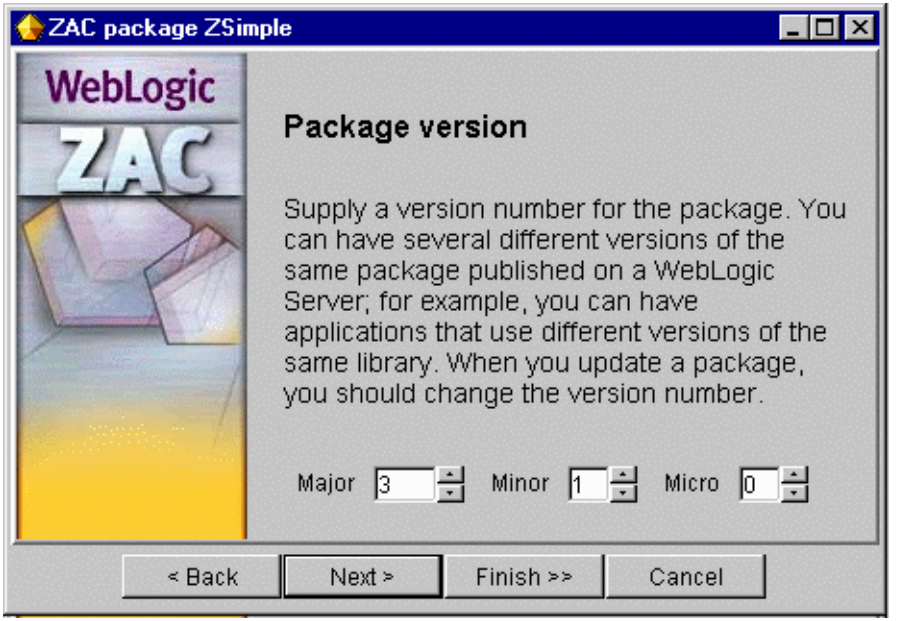
4. Supply a title for the package and a short prose description of it. Both will be used for display purposes.

Figure 2-4 Supplying a Title and Description



5. Set the version number.

Figure 2-5 Setting the Version Number



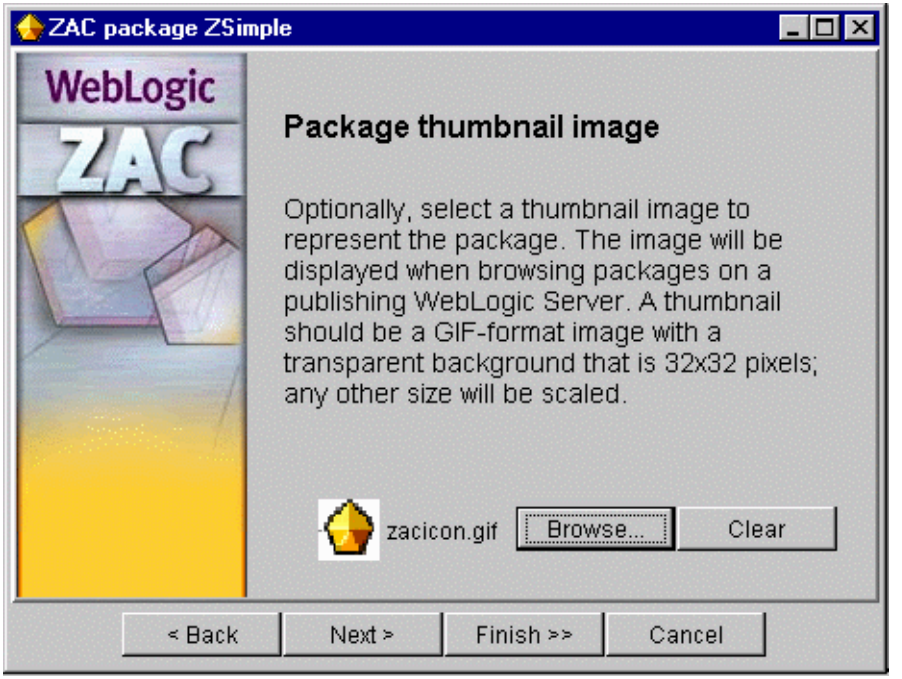
6. Browse to the directory that contains the files for the ZAC package. Everything in the directory you choose, plus everything in all its subdirectories, will be published.

   You can exclude files from the published package by adding to the list of file types to be ignored. This allows you to locate files that may be used to generate the package in the same directory, without publishing them. In general, you will always want to ignore index.xml and index.osd files, which are specific to each client. A list of suggested files to exclude is supplied; you can add to or delete from this list for your published package.

Figure 2-6 Choosing the Top-Level Directory of the Package

7. Find a GIF image to use as a thumbnail for the package. The image should be 32x32 pixels with a transparent background for best results.

Figure 2-7 Supplying a Thumbnail Image



8. If you are publishing an applet, you will be asked to enter the applet's main class, the CODEBASE, and a list of applet parameters that would customarily be listed in PARAM tags. If the applet is online, you can enter a URL that ZAC will use to find the applet and complete the list of parameters automatically. To identify the applet, supply:

   The main classname

   If you enter a URL for an online location of the applet (along with the document base), ZAC will find the applet and supply its parameter list automatically.

   The document base

   This is the URL of the webserver that is hosting the applet. Supplying the document base and the name of the class is enough information for ZAC to find an online running version of the applet and load the parameter list automatically.

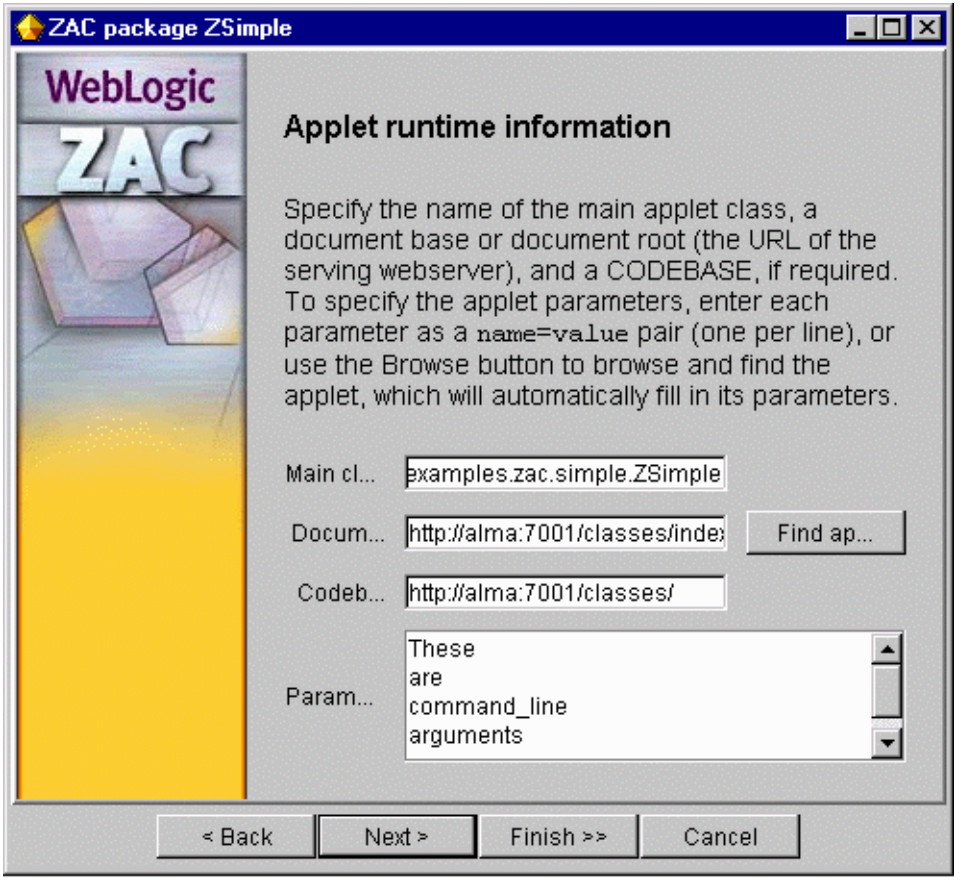
The CODEBASE attribute

This is a URL where the applet may retrieve additional class files as required. It is a common practice to set this to `http://yourserver:port/classes` where `yourserver` is your WebLogic Server and you have registered the ClasspathServlet against the virtual name `classes`. For more details see the Administrators Guide on Registering the WebLogic servlets.

The applet parameters

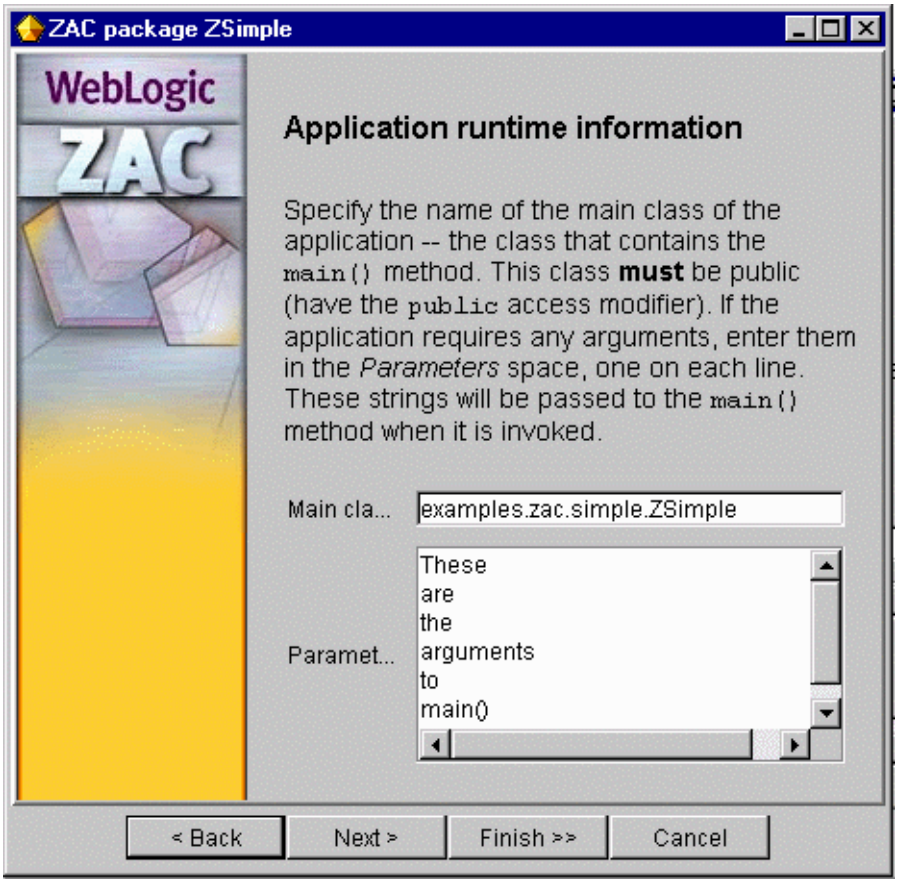
Applet parameters are a list of name=value pairs that supply initialization and runtime variables to the applet, analogous to those supplied between the <APPLET> tags in an HTML file.

Figure 2-8 Specifying Applet Parameters



9. Enter the path to the appropriate class for the published package. If you are publishing an application, you will be asked to enter the full path to the Java class that contains the `main()` method that starts the application. You may also enter any initialization arguments, required by the `main()` method, in the Parameters text area.

Figure 2-9 Locating the Class to Start the Applicatio*n*



Specify the CLASSPATH for your package (Required). If all of the necessary classes are contained in the published package itself, you can simply specify a dot "." for the current directory. The CLASSPATH is always relative to the top-level directory for the application. If you are publishing an application, the class that contains the `main()` method to run your application must be in the CLASSPATH of the package.

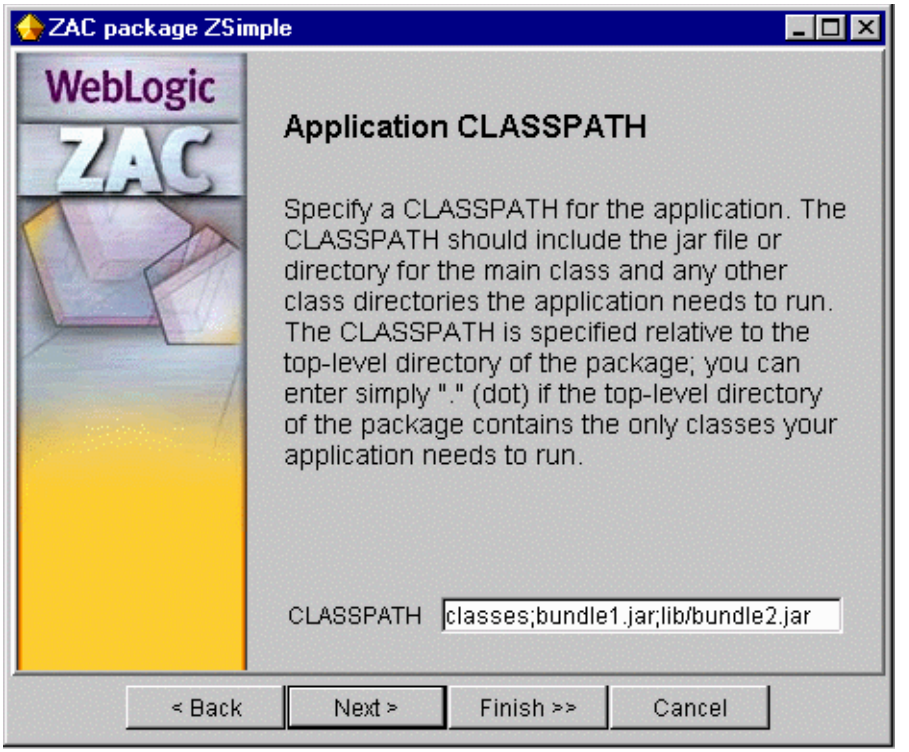
When publishing any type of package, CLASSPATH entries are always relative to the top of the local directory being published. For example, when publishing the local directory `c:\myapps`—which contains the following—

```
c:\myapps\classes\foo\Main.class
c:\myapps\bundle1.jar
c:\myapps\lib\bundle2.jar
```

where the class `foo.Main` is contained in the package `foo`—you would set the CLASSPATH for the package as:

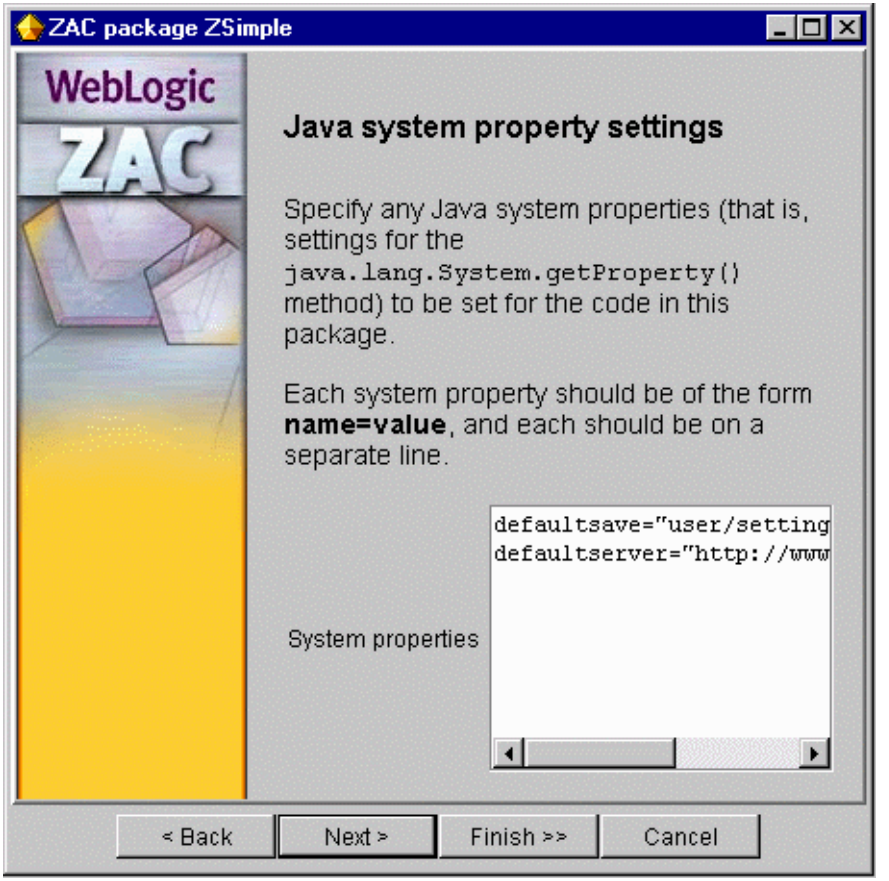
```
classes;bundle1.jar;lib\bundle2.jar
```

Figure 2-10 Specifying the Package CLASSPATH



10. Here, you may specify Java system properties that your application requires. List one **name=value** pair per line.

Figure 2-11 Specifying Java System Properties



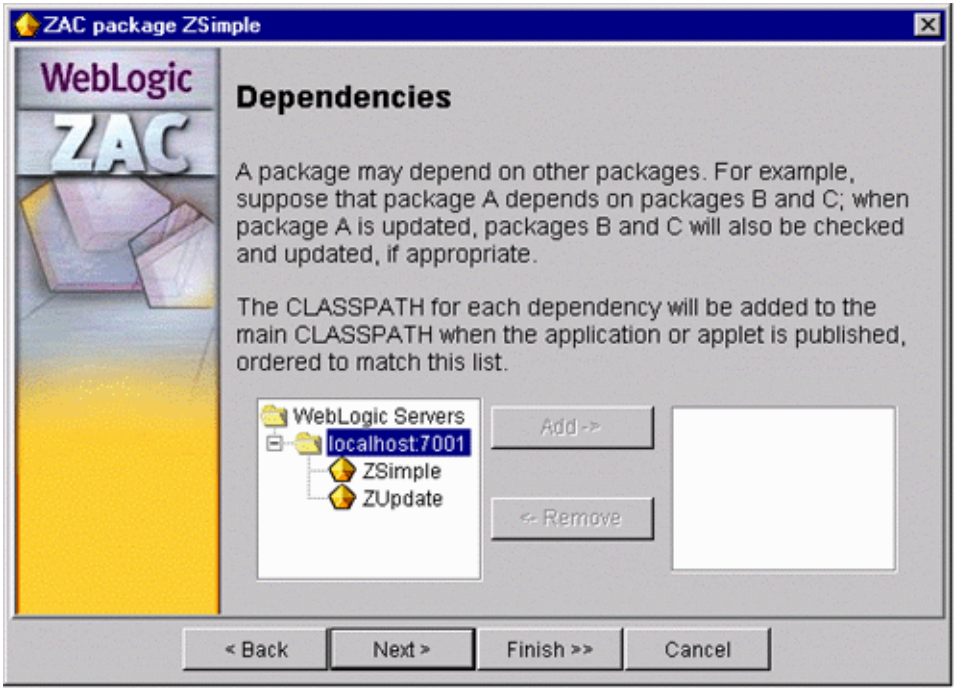
11. Next, you may set up dependencies for your package upon other packages. ZAC will ensure those packages are installed and up-to-date on the client machine also. You must make ZAC packages of shared libraries or other applications that your package depends upon, and specify the dependencies using package names.

    Using dependencies with ZAC allows several different applications to share common code on the client machine. When common code is updated, it is consequently updated for all dependent ZAC applications.

    For example, if you were developing applications that depend upon the WebLogic and the Swing classes as libraries, you could list these as

dependencies for each application, and there need only be a single copy of those libraries on the client machine.

Figure 2-12 Setting up Dependencies



Locate a published package from an available WebLogic Server in the left hand window and click 'Add'. Details about the depended-upon package are displayed in the right hand window. To remove a depended-upon package, select it in the right hand window, and click on 'Remove'.

Publishing a ZAC Package

Once you have created a new package or updated details for a package that you have published previously, you are ready to publish the package to a WebLogic Server.

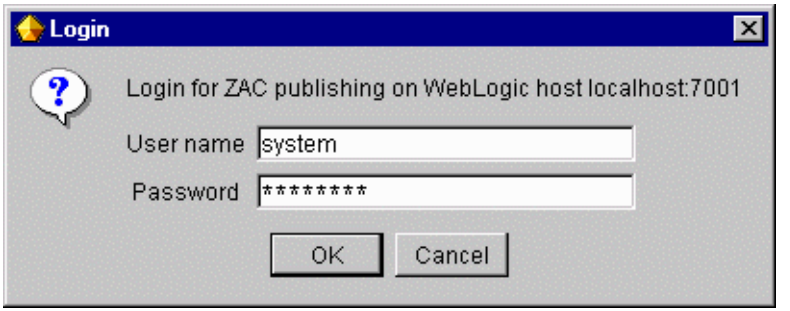
1. Select the package you wish to publish in the ZAC Publish Wizard main window. In the *"Package"* menu, select **"Publish"** if it is enabled. If you have multiple WebLogic Servers listed in the 'WebLogic Servers' panel, the second option **"Publish to"** will be enabled, which allows you to select the WebLogic Server to publish to.

   If you select *"Publish"* the WebLogic Server is automatically chosen for you. This will be either the only server listed, or the last server that you published to or reverted from.

   If you select *"Publish to"*, you will need to select the address/port of the WebLogic Server to which you want to publish this package.

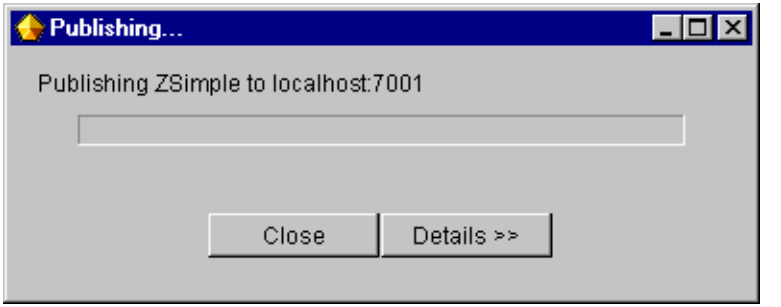
2. The first time you publish to a WebLogic Server, and each time you publish after you have restarted ZAC, you will be asked to supply a name and password. To publish, you must supply a username and password for the T3User that has "write" permission for the ACL `weblogic.zac` in the `weblogic.properties` file. If unset, this ACL defaults to granting only the "system" user with write (publish) privileges.

Figure 2-13 Supplying Authorization for Publishing



3. The progress of the publishing operation is shown in the "Publishing..." window. When complete, press **"Close"**, or **"Details >>"** to review the details of the published package.

Figure 2-14 The Publishing Progress Dialog



Reverting a ZAC Package

If you make changes to a local ZAC package before publishing it, you may revert the package to a version previously published on a WebLogic Server. Select the local package, and choose **"Revert"**, or **"Revert from"** in the **"Package"** menu. If you are running more than one server you will need to select the later option in order to choose the server to revert from. If you choose **"Revert"**, the server is chosen for you as either the only server that is listed, or the last server that you published to or reverted from.

Using the command line Publish Utility

You can use the command line Publish Utility to publish a ZAC package on the WebLogic Server, as an alternative to using the ZAC Publish Wizard. The Publish Utility is a stand alone java application that you may run from the command line, or invoke from a shell script. You configure the actions of the Publish Utility by supplying command line options. These options closely follow the parameters that you define in the ZAC Publish Wizard. Use the Publish Utility on the command line as follows:

```
$ java weblogic.drp.admin.Publish [options]
```

-name zacPackage

(Required) The name for the ZAC package you are publishing, as it shall appear on the WebLogic Server. Note that the name should not contain any spaces.

-dir packageDir

(Required) The pathname of the top-level local directory that contains the entire contents for the ZAC package that you are creating and publishing. The contents of the directory, and all subdirectories are included in the new package.

-host hostname

(Optional) The host name of the WebLogic Server you are publishing to. This defaults to *"localhost"*.

-port portnumber

(Optional) The port number of the WebLogic Server you are publishing to. This defaults to *"7001"*.

-login username *-password* passwd

(Optional) You must specify a username and password to publish a package on a WebLogic Server that uses security controls for publish authentication. By default, this is set to the *system* user and password by the WebLogic Server. You may grant publish *(write)* privileges to a user or group by specifying an ACL in the `weblogic.properties` file. See Setting up WebLogic for Publishing with ZAC (step 2) for details.

By default, the Publish Utility will attempt to publish the package without using a username and password. This will fail unless ZAC publish `write` privileges have been granted to `everyone`.

-verbose | *-v*

Causes the Publish Utility to print verbose messages about its operation.

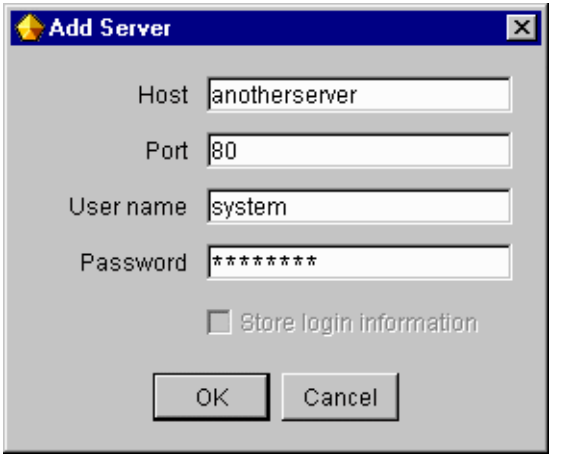
-help

Prints a short summary of usage for the Publish Utility.

Connecting the Publish Wizard to Other Servers

When you start the Publish Wizard, it will connect automatically to the WebLogic Server running at the default location, and display it in the lower window pane. To discover other WebLogic Servers, select **"Add"** from the **"server"** menu to access the following dialog.

Figure 2-15 The Add Server dialog



Updating a Published ZAC Package

Updating a previously published application is simple.

1. Start the Publish Wizard.
2. Select the package you want to update
3. Select **"Open"** from the **"Package"** menu.

4. Carry out the same steps as for creating and publishing a new package.

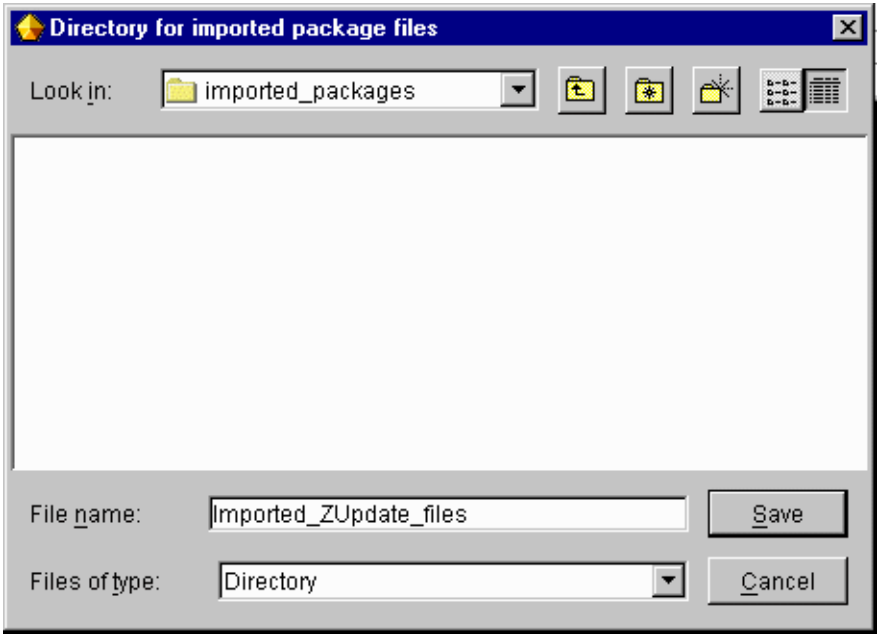
Importing a Published ZAC Package from Another Server

You can import a published package from one WebLogic Server and publish it on another WebLogic Server. At that point, the package becomes a separate copy at will not be updated if the original package is updated.

To import a published package:

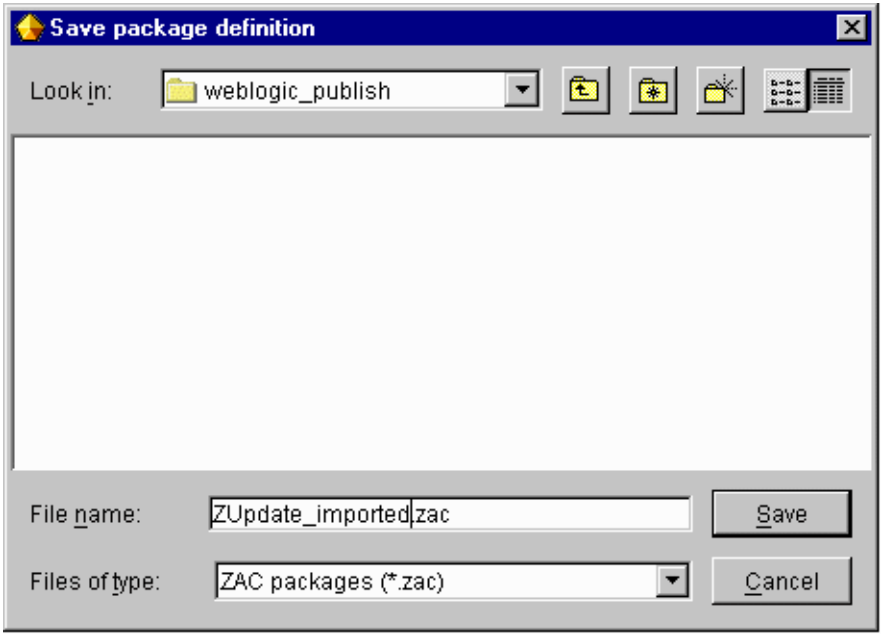
1. Add the WebLogic Server host you wish to import from to the Publish Wizard's host list. See "Connecting the Publish Wizard to Other Servers" on page 2-31.
2. Select the published package in the other WebLogic Server host.
3. Select **"import"** from the **"Package"** menu.
4. You will be asked where you wish to store the imported ZAC package files. Select an appropriate directory, and enter a directory name in the text box. A new directory will be created under that name in the current directory you have browsed to. If you do not specify a name, the operation will not be successful.

Figure 2-16 Storing the ZAC Packaged Files



5. Next, you are prompted for a name to save the ZAC package definition under. This should be saved in the default directory where you keep your ZAC package definitions. This is usually saved under the directory `/weblogic_publish`.

Figure 2-17 Saving the ZAC Package Definition



6. The imported package will apper under the “local package” list, under the name it was published with on the originating server.
7. You may now publish the package, as described under Publishing a ZAC Package.

Removing a Published ZAC Package

The remove operation only removes the published application files themselves, including directory for the package that is stored in a package directory in the ZAC publish root.

Removing a package does not affect the WebLogic Server or the original files from which the package was published, nor will it remove local files from a ZAC client.

1. From the Server menu, choose *Remove package*.

2. Browse the WebLogic Server from which the package is to be removed.
3. Select the package and press the *Choose* button.

Creating an Installer/Bootstrap Application

You can also use the ZAC Publish Wizard to create a set of native programs — an installer and a bootstrap — for various operating systems that become part of a published Java application.

The installer program is a native executable that installs your published Java program on the local machine; it may also install a JRE. It doesn't require a Java environment itself, so it can run out-of-the-box in the native OS. It's a little like an InstallShield for Java.

The bootstrap is also a native program; the user runs the bootstrap to invoke the published application. The bootstrap takes care of monitoring for updates, downloading and updating the user's application, and other administrative ZAC functions.

Installer/bootstrap programs can be created for the following OS types:

- Win32 (Windows95/98 and Windows NT)
- Solaris/SPARC
- Linux/x86
- DECUnix/Alpha
- HPUX (HPUX 11)

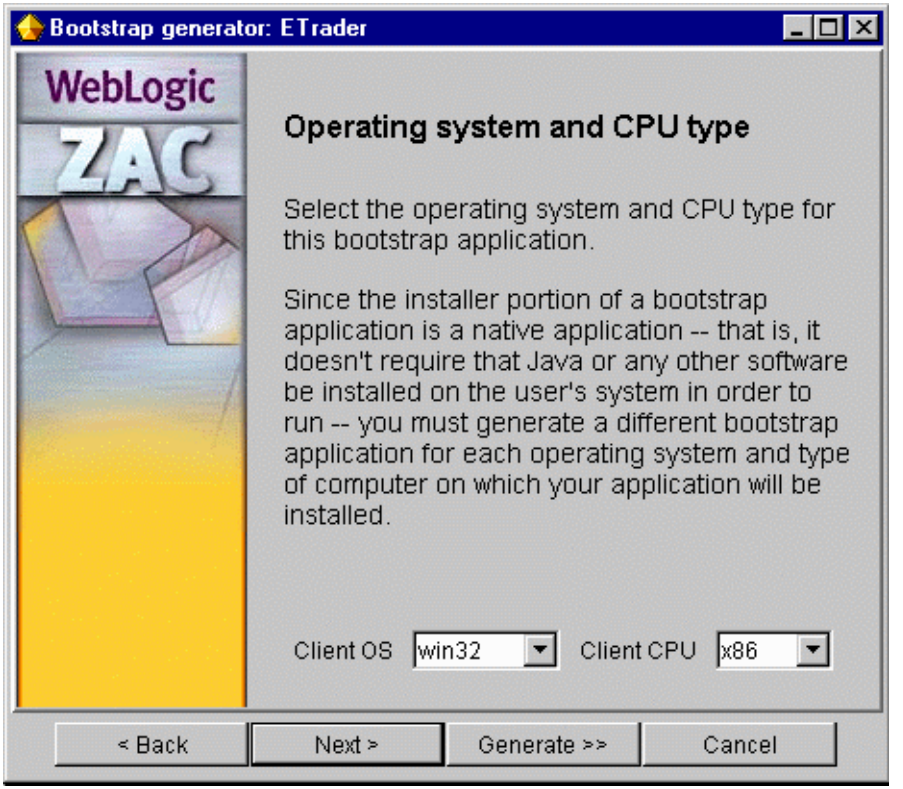
Both the installer and the bootstrapper are small native, applications. You will need to create these for each type of operating system and CPU type that you expect your clients will use.

To create the installer/bootstrap executables in the Publish Wizard:

1. Start the Publish Wizard, and highlight the ZAC package for which you wish to create a bootstrapper installation program. You must have previously created a ZAC package for your application and published it on the WebLogic Server.

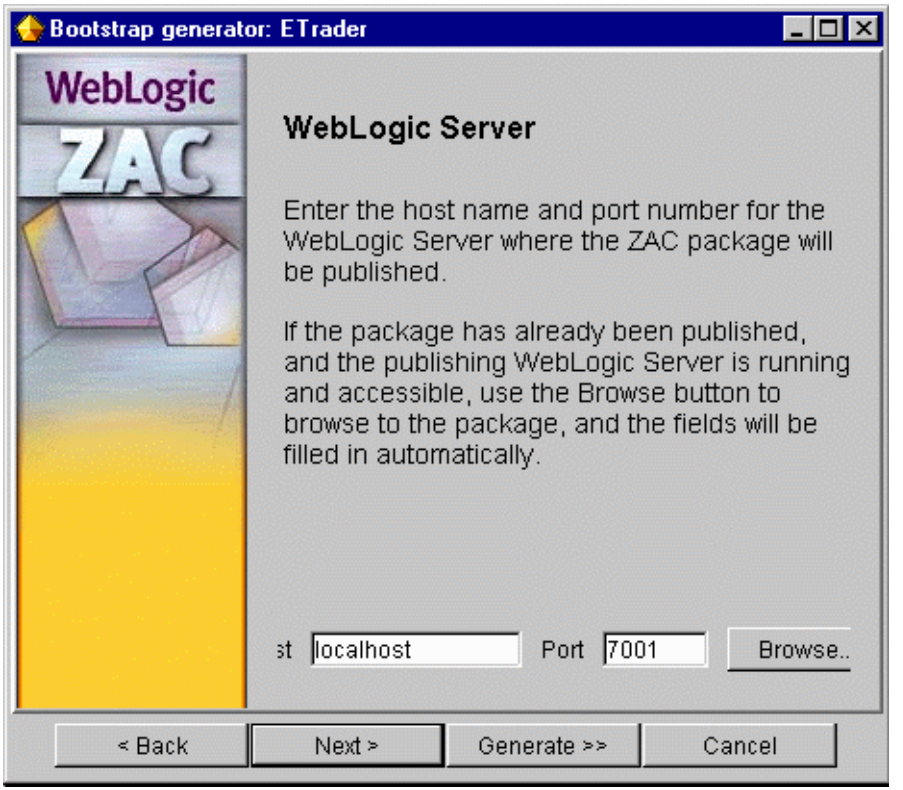
2. Select Create bootstrap app... from the Package menu.
3. Set the appropriate operating system and CPU type for this application. Choosing an OS and CPU will set some default values that you can adjust as necessary.

Figure 2-18 Setting OS Details for the Bootstrap



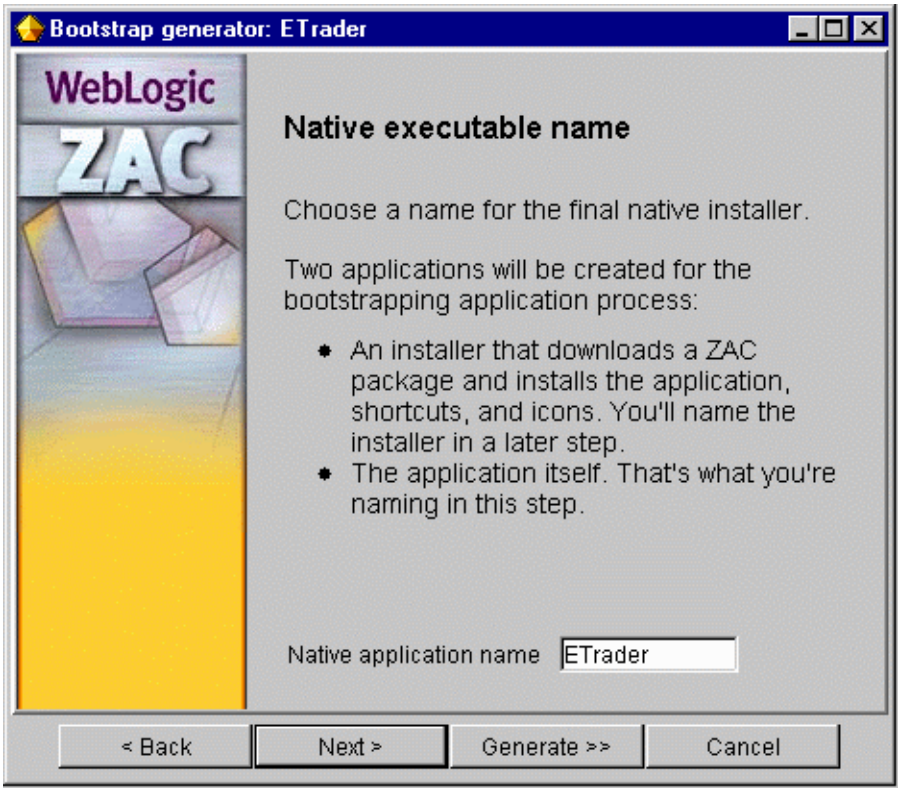
4. Enter the host and port of the publishing WebLogic Server. If you are preparing a bootstrapper application for a package installed on another server, you can find the available servers by pressing the *Browse* button.

Figure 2-19 Identifying the Publishing WebLogic Server



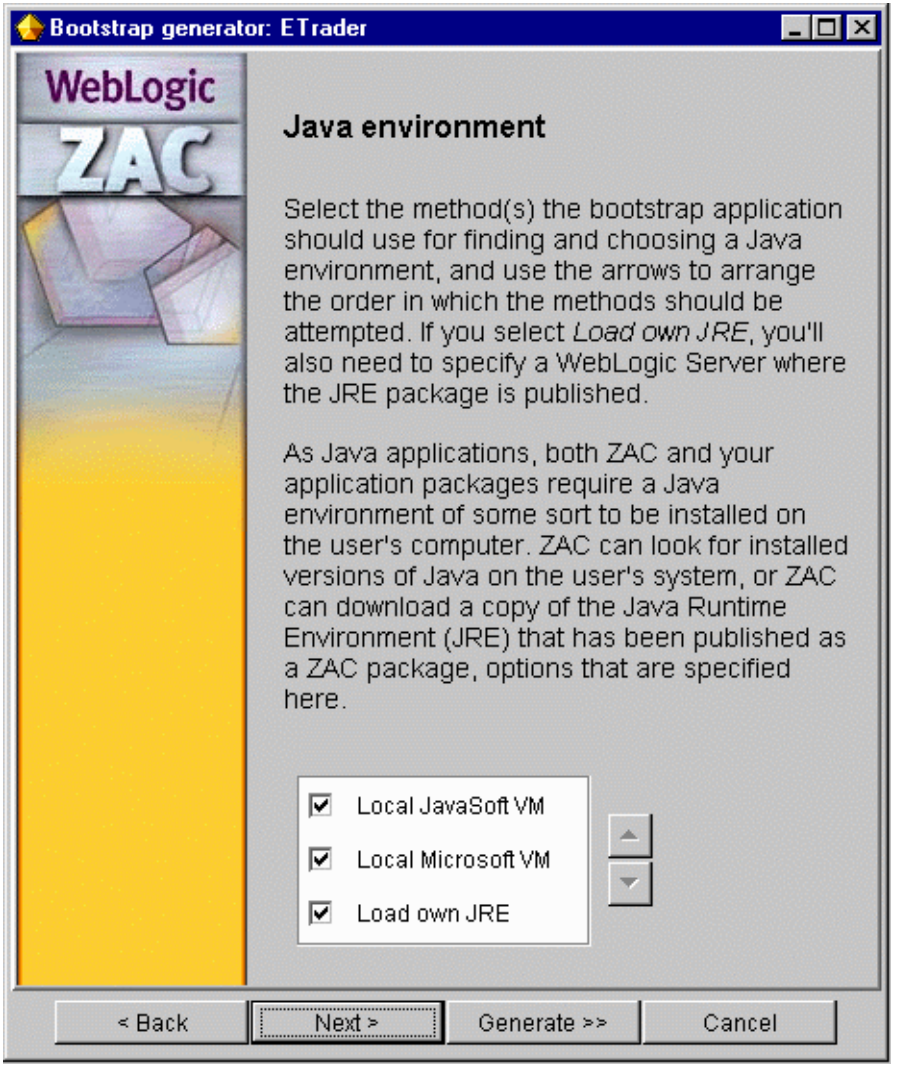
5. Assign a name for the native bootstrapper executable. The bootstrapping process creates two applications: the installer package (usually very small) that the client downloads and runs initially, and the bootstrap that the user uses to invoke the published application each time he runs it. What you name in this step is the bootstrapper executable.

Figure 2-20 Assigning a Name



6. Select one or more methods for finding and choosing a Java environment. You may depend on a locally available copy of the Microsoft or JavaSoft JVM, or you can load a JRE (Java Runtime Environment) that has been published as a ZAC package. Check the options that you want to be available to the client, and then order how those options should be processed by using the up-and-down arrows to the right.

Figure 2-21 Making the JRE Available to the Client Application



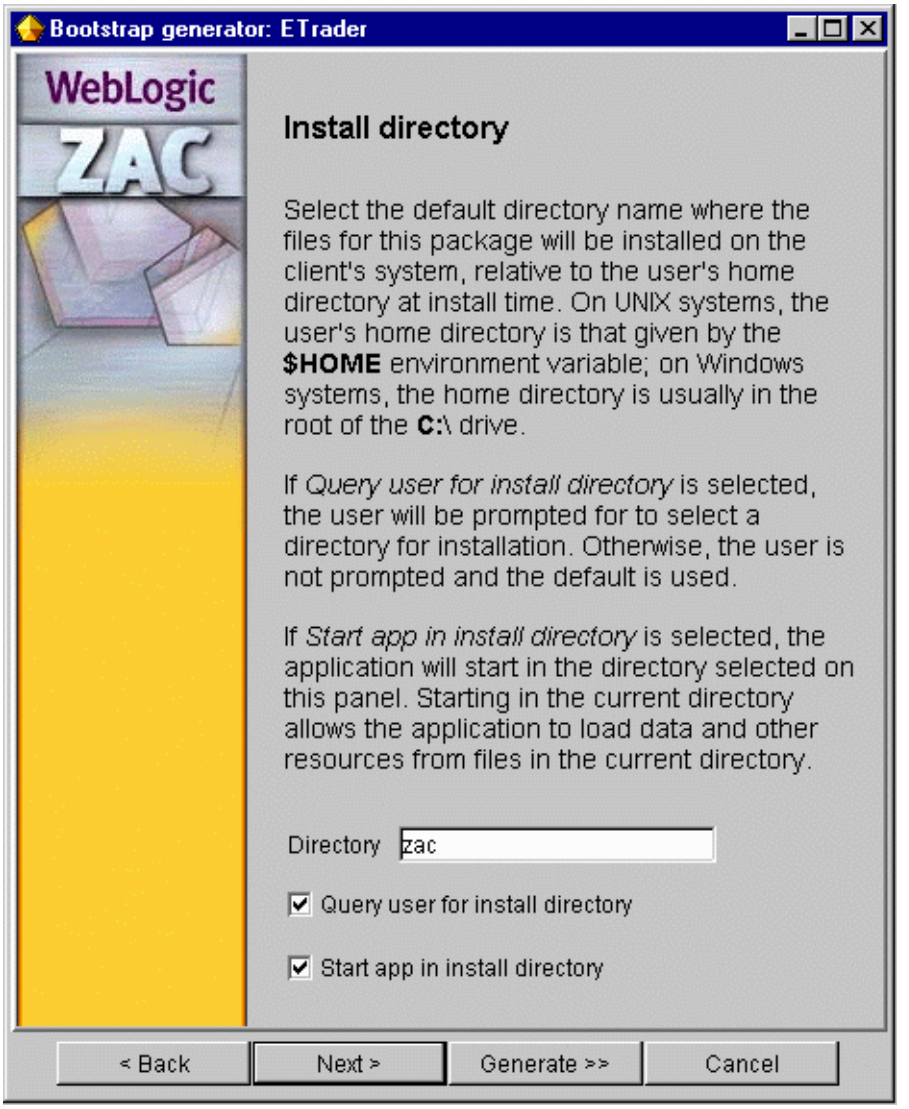
7. If you selected Load own JRE, you will be prompted to locate the publishing WebLogic Server where the installer can find a published package of the JRE. For details on packaging your own JRE with ZAC, see the section Packaging a JRE later in this document.

8. Specify options for the Sun VM memory flags when if is initiated by the bootstrap application. The default values are specified here. You should set them accordingly if you application has special needs.

Figure 2-22 Setting Memory Options for the Sun VM

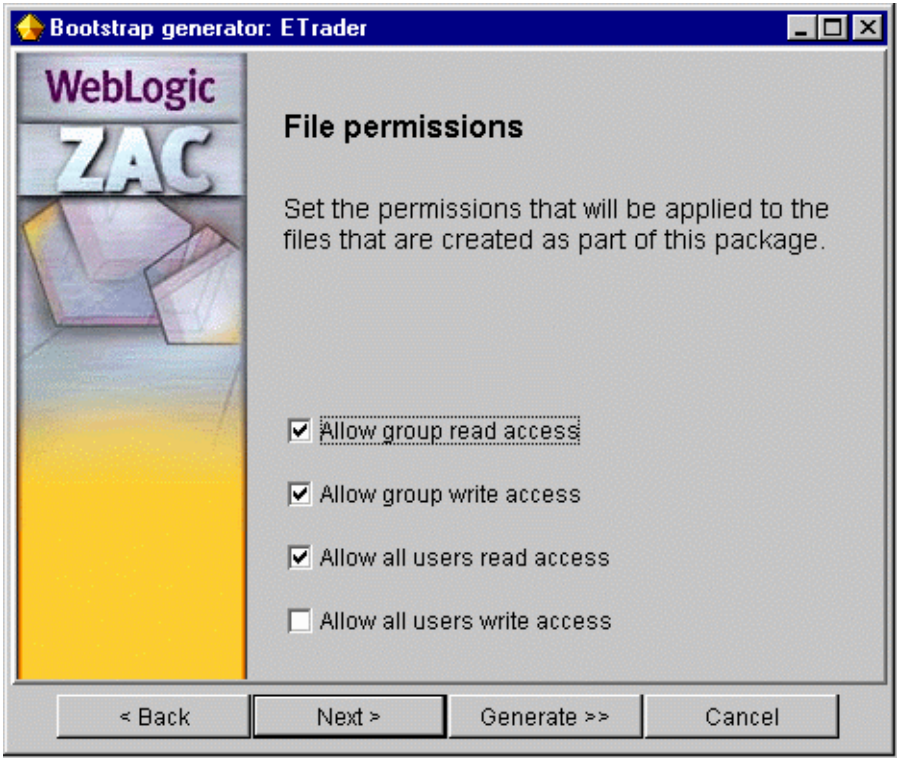
9. Enter the local directory on the client machine into which the bootstrap application should be installed when the user double-clicks the .exe file. The directory will be absolute if you begin the path with a slash (forward or backward depending upon the operating system for which you are publishing this bootstrap). If you choose the *Start app in install directory* option, the directory selected here is where the application will start from.

Figure 2-23 Setting a Local Client Directory



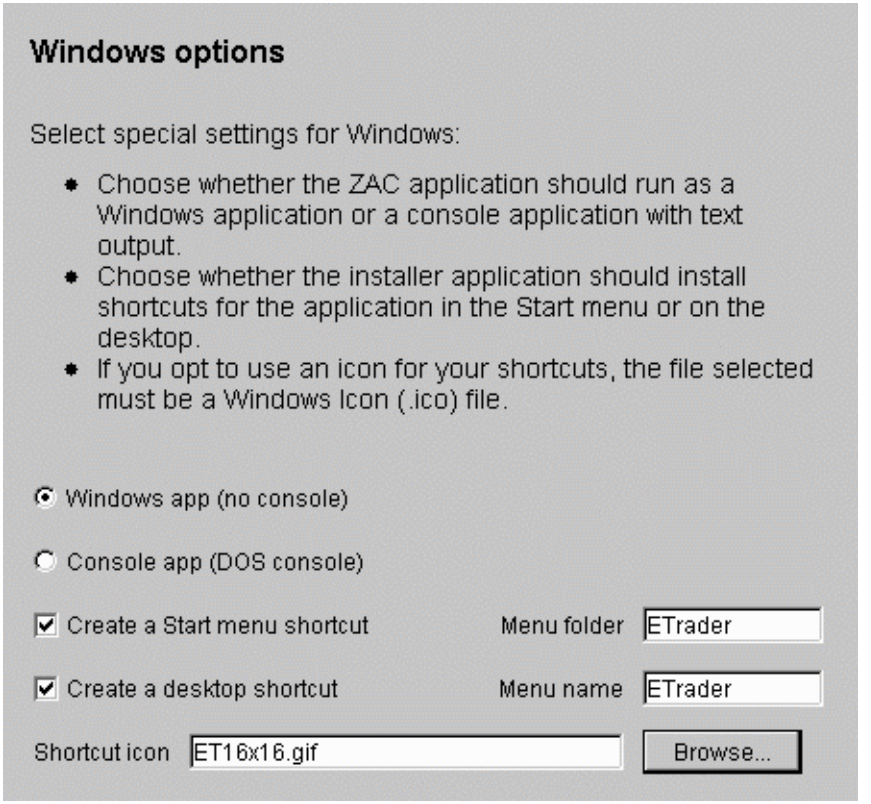
10. Set the client permissions for the files associated with this package.

Figure 2-24 Setting Access Permissions for Client Files



11. If you are publishing for Windows, you will be asked to choose some special settings for Windows. You can install shortcuts for the Windows Start menu or desktop and you can set an icon for the bootstrapper package.

Figure 2-25 Setting Windows-Specific Options



12. Specify the behavior of the ZAC installation-bootstrap and the post-installation bootstrap. These options are defined as follows:

Update application

If this option is checked, the bootstrap executable will check for newly published versions of the ZAC application and update it if necessary. You may wish to disable this feature if one of these conditions is true:

- You wish the client to run the application offline
- You have embedded ZAC update functionality directly into the application using the ZAC API
- You plan to generate another bootstrap executable for the purpose of updating the ZAC application (See *Launch application* option below)

Check all dependencies

If this option is checked, the bootstrap executable will check for newly published versions of the ZAC packages that this application depends upon. You may wish to disable this feature for similar reasons to those listed above.

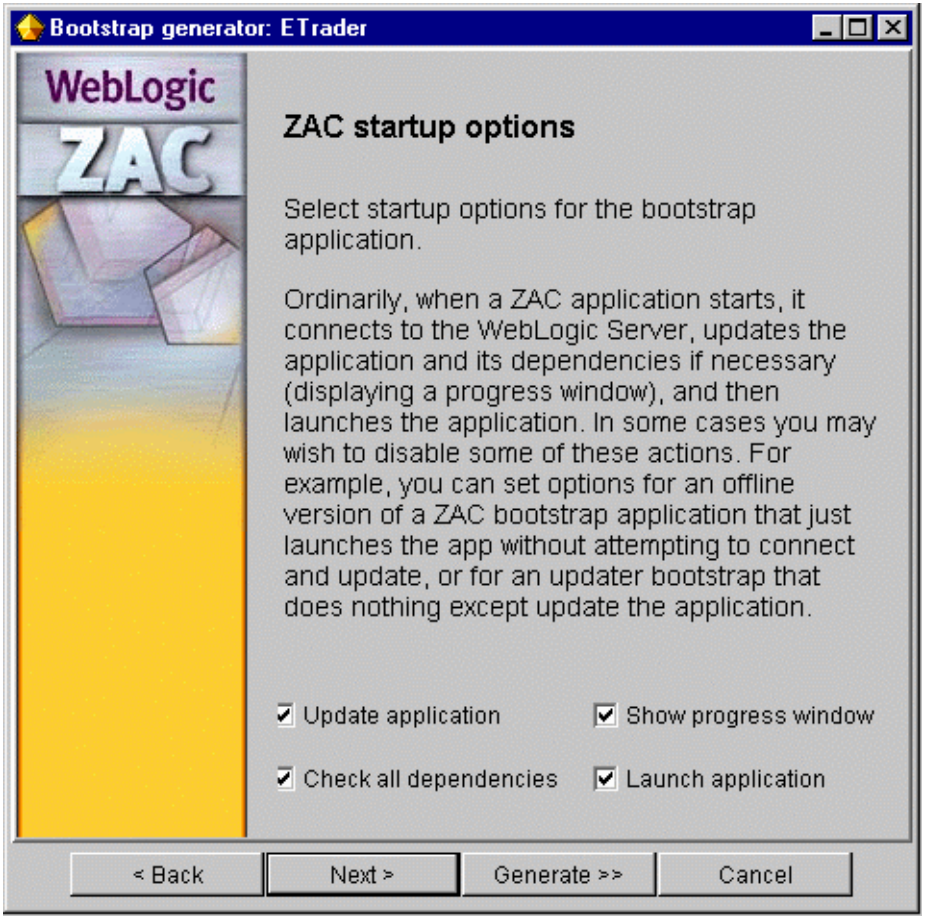
Show progress window

If this option is checked, the bootstrap executable will display a meter indicating the progress of the download when the application updates. If you wish the update to be silent, you can uncheck this option.

Launch application

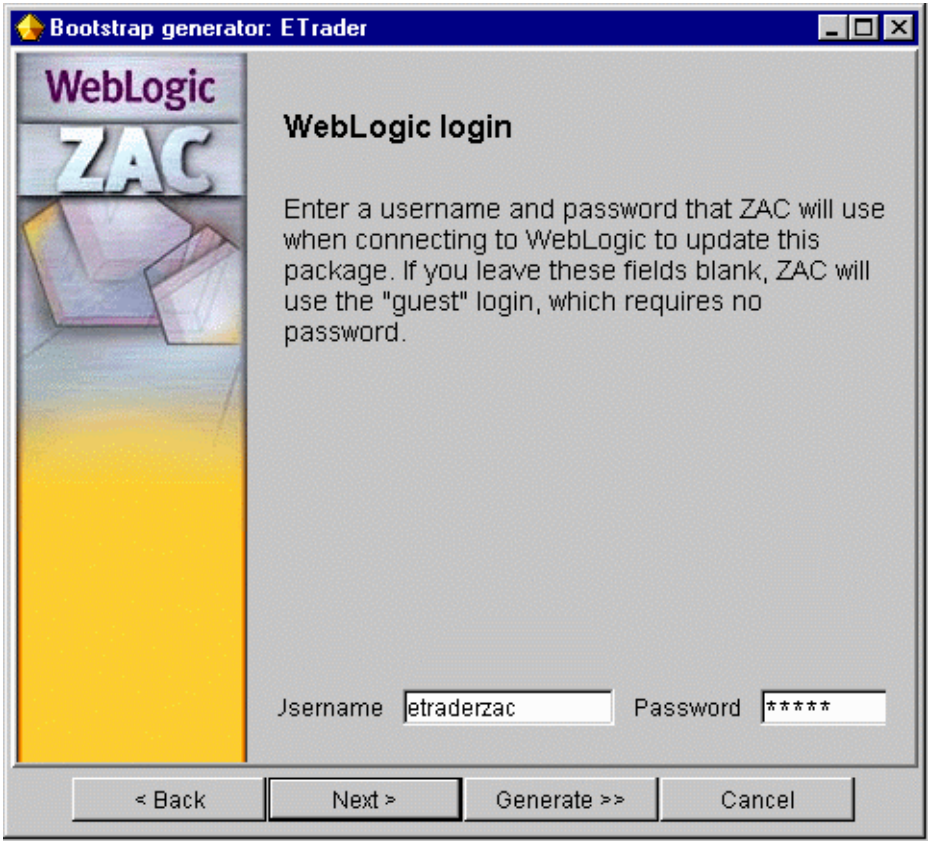
This option is usually checked. If this option is *not* checked, then the bootstrap executable will *not* launch the application. You might uncheck this option to create a bootstrap that will only update the application on the client machine. You could use such a bootstrap in conjunction with a bootstrap that only starts the application and does not update it.

Figure 2-26 Setting up Bootstrapper Options



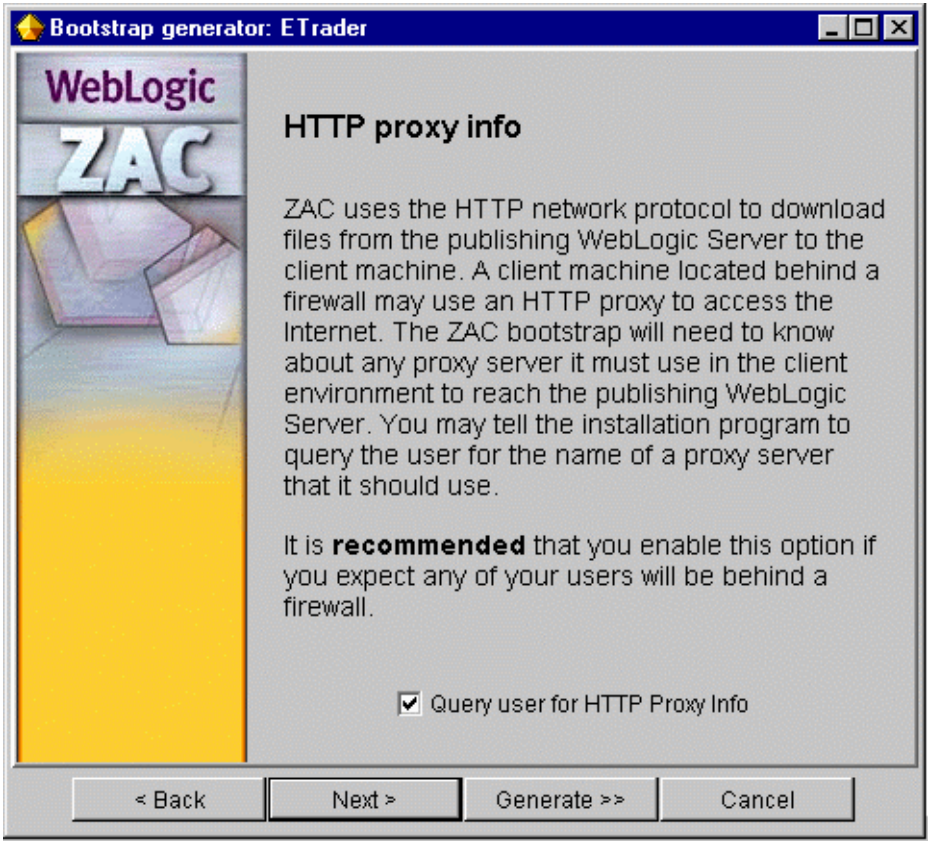
13. Set a WebLogic username and password for access to this package. This should correspond to a WebLogic user that is on the access control list (ACL) for read permission to this application on the publishing WebLogic Server. Note that if you do not set an ACL for a published package, the permission to read defaults to the special group everyone and the permission to write (publish) the package defaults to the special administrative user system. That means that anyone can download your published package, but only a system-level user can publish or republish packages.

Figure 2-27 Setting the User and Password for Secure ZAC Packages



14. If you expect your users to access the internet from behind a firewall, you should configure the bootstrap to ask for an HTTP proxy server through which it may access the WebLogic Server. It is a good idea to always leave this option checked, unless you know that your clients are not behind a firewall.

Figure 2-28 Configuring the Bootstrapper to Prompt for HTTP Proxy Details

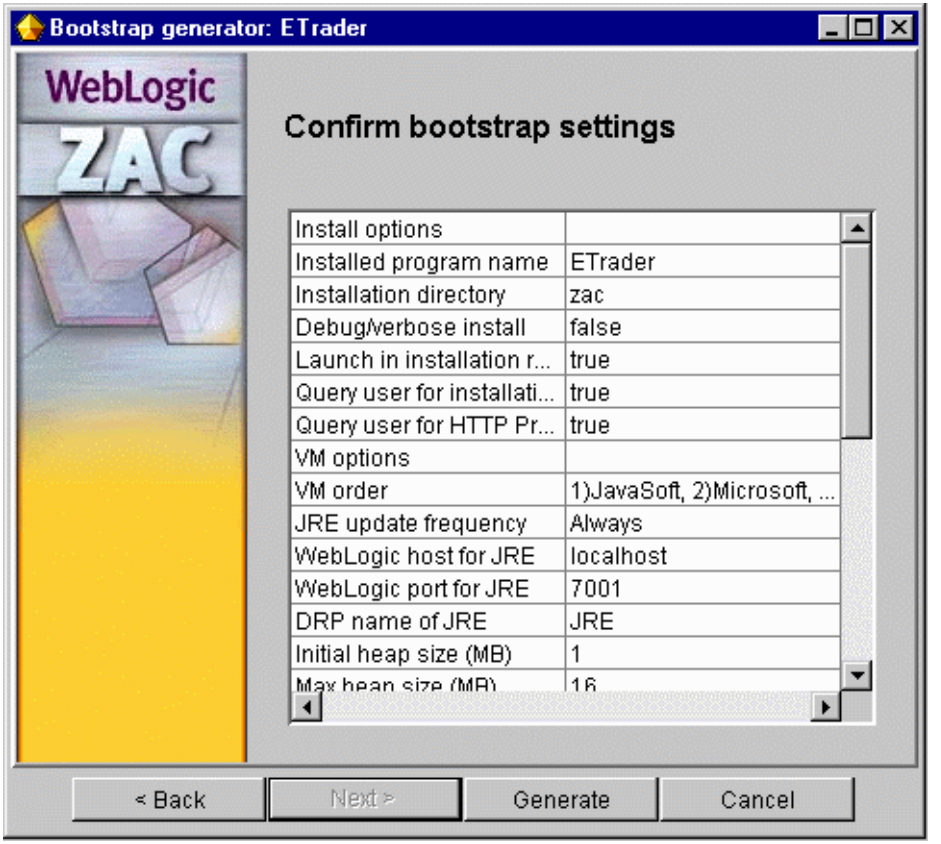


15. Set the debug mode and verbosity of the application. This is useful while you are testing the deployment of your ZAC application from a client machine.

Figure 2-29 Setting Debug Mode

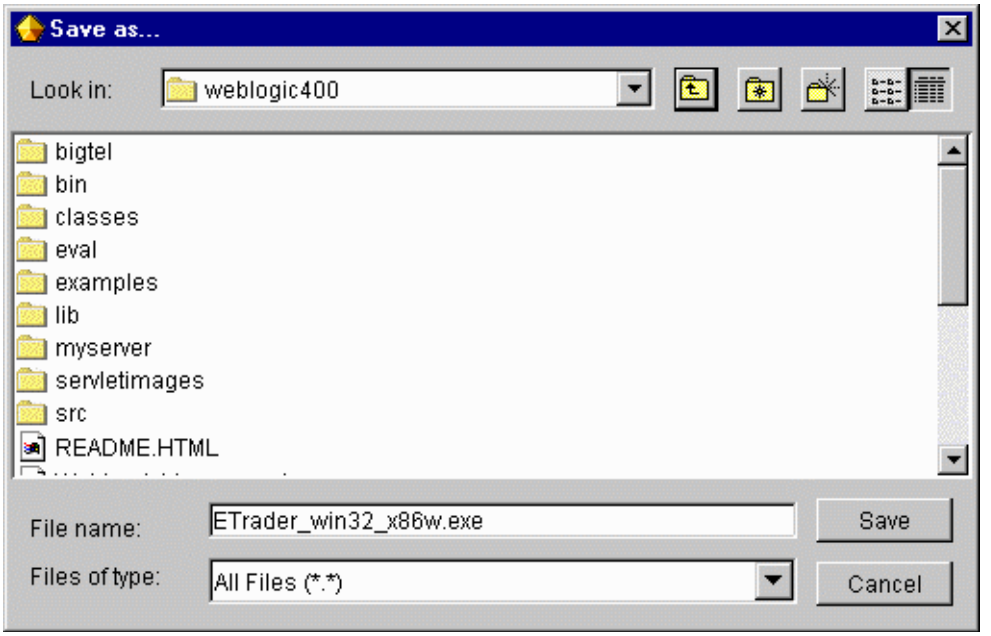
16. Review the bootstrap settings. Press the *Generate* button when completed.

Figure 2-30 Confirming Your Choices



17. From the Save As... window, select the location and name of the installation executable. The default name is based on the ZAC package name and the target OS; but you may call this executable file anything you like. You will deploy this executable to clients to install your application.

Figure 2-31 Saving the Generated ZAC Installation Program



Packaging a JRE

You can wrap the entire JRE in a ZAC package and include it with your ZAC application. Including the JRE means that you do not need to make any assumptions about the end user's machine and you are assured that everything necessary for your application is provided, including the correct version of Java. The downside is that the initial download package is larger, and it may install the JRE even though there being another JRE present. This small initial inconvenience may be more desirable than causing installation problems for the user.

Before You Begin

The following steps describe how you should make a ZAC package of theJRE. You can then include this ZAC package as a dependency of yourapplication. Before you start the Publish Wizard, you should do thefollowing:

1. Download the JRE install-shield. The JRE from JavaSoft can be downloaded from JavaSoft.
2. Install it on your development machine. In these instructions, we'll assume you have installed it in the directory `c:\jre117`.

Now start the Publish Wizard and continue the process.

Creating and Publishing a ZAC JRE Package.

1. Create the package type as "Library" and give it CLASSPATH "." (CLASSPATH doesn't matter in this case). Give it no dependencies.
2. Create a new package for the JRE. We recommend that you choose a name that reflects the OS and CPU type; for example, "JRE_117_win32_x86".
3. Enter a title and description for the JRE package. You can enter any title you wish.
4. Enter the package version as closely to the JRE version as possible. For example,
   For JRE 1.1.7 enter '1' '1' '7'
   For JRE 1.2.0 enter '1' '2' '0'

Entering the correct JRE version is important since the version string is used to determine the default local installation directory for the JRE. Using the same version number as the JRE version will minimize the chance that two separate ZAC packages will overwrite each other's JREs with incompatible versions. Of course, this is only used to determine the default install directory; the user ultimately decides where the JRE shall be installed.

5. You may optionally select a thumbnail image for the package.
6. Specify the top-level directory where you have previously installed the JRE. (In this example, we're using `c:\jre117`.)

7. Specify simply . (dot) as the CLASSPATH for your library.
8. Skip the other dialog screens and publish the package. Make sure that the publishing server is running before you try to publish.

After the publishing step is complete, you should see the new ZAC JRE package appear in the 'ZAC Publish Wizard' window.

Specifying a Published JRE Package for an Application

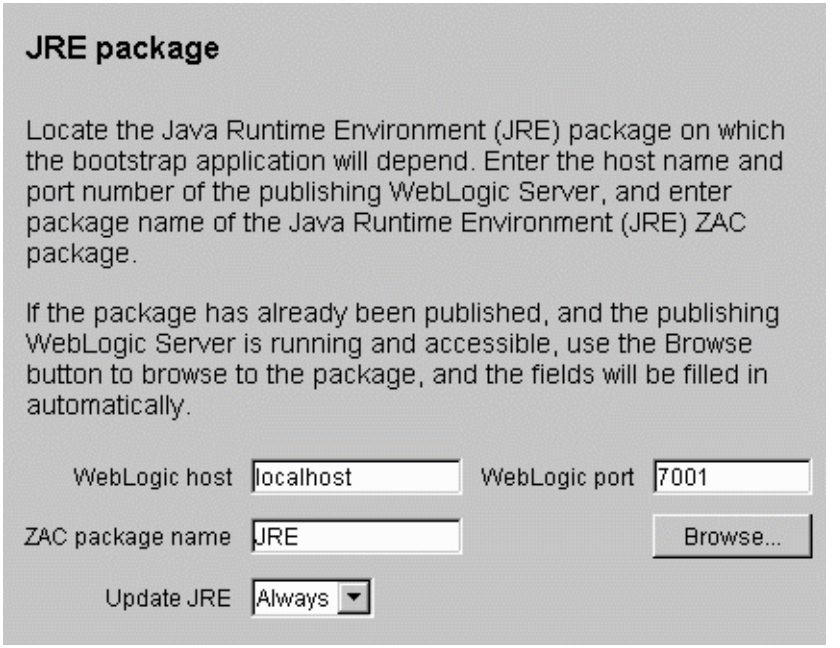
Now that you have created your own JRE ZAC package (see section above), you need to add a dependency between it and your ZAC application. Here is how:

1. In the Publish Wizard, create an installer/bootstrap executable for your application, using the instructions in this document in the section Creating an Installer/Bootstrap Application.

2. During the create process, when you make choices on how to configure the Java Environment, you will see three choices that show the preference for choosing a JRE on the client user's host. You can order, select, or deselect these choices as illustrated on the right. Select Load own JRE as one of your VM choices and adjust its position in the preference list using the arrow buttons; then press the Next button.
3. When the bootstrap wizard asks where the JRE is published, provide the following:
   - The hostname and port for the WebLogic Server where you published the ZAC JRE
   - The ZAC name (e.g., "JRE_117_win32_x86") under which the JRE distribution is published

You can type in the path or Browse for the published package on the appropriate server in your server list.

Figure 2-32 Locating a Published JRE



4. Press the Generate button, or proceed to the end of the bootstrap wizard.

5. Finally, choose a file name and path to save the installation executable on the publishing server. For example, `\weblogic\public_html\MyAppInstall.exe` saves the installer/bootstrapper into the default document root of a WebLogic Server host, where you can publish access to it from an HTML page.

   The native installer will be small, probably about 250 KB for a Windows architecture. You can distribute it to clients however you like; for example, put a link to it on a web page, or attach it to an email.

Debugging and Testing a Published Application

The WebLogic distribution contains native executables for running the ZAC bootstrap from the command line for Win32, Solaris, Linux, and DECUnix. The Windows version (`zac.exe`) is located in the `bin/` directory; the non-Windows versions are located in the `lib/{arch}/zac_{arch}/` directories. Each executable also has an `_g` version that can be used for verbose debugging.

Use the appropriate version of the command line bootstrapper to install or run a ZAC package. For example, you run the Windows version as shown here:

```
$ zac.exe -name zacPackage options
```

The options are defined as follows:

-name zacPackage

Required. The name of the application to launch or download. For example:

```
$ zac.exe -name ETrader
```

-host hostname

Hostname of the publishing WebLogic Server. This defaults to `localhost`. For example:

```
$ zac.exe -name ETrader -host zac.weblogic.com
```

-port port

Port at which the publishing WebLogic Server is listening for login requests. Defaults to `7001`. For example:

```
$ zac.exe -name ETrader -host zac.weblogic.com -port 80
```

-proxy

Prompts for proxy information, according to the configuration set in the ZAC bootstrap wizard when the package was published.

-dir localDir

The local (client) directory where the ZAC application files are located. This defaults to the current working directory. The following example launches a ZAC application called *ETrader*:

```
$ zac.exe -name ETrader -dir /usr/local/zac/
```

For this example, all files for the ETrader application and all of its dependencies will be stored in the subdirectories below `/usr/local/zac/`.

The actual ZAC application ETrader is stored beneath this directory in another directory with the same name as the package, that is `/usr/local/zac/ETrader/`. The ZAC bootstrapper looks inside this directory for the OSD application manifest (`index.osd`).

-root

Start the Java ZAC application in the ZAC root directory. The default it to start the ZAC application in the current working directory.

-vm JVM type

Specifies the order of preference for locating a JVM on the client's machine. Specify any combination of the characters "S","M",or "O" where the highest preference is on the left, and the characters represent:

- S (Sun Java VM)
- M (Microsoft VM)
- O (Own VM. You provide your own JVM as a published ZAC package.)

If you use "O" in this argument, you must specify a JRE with the *-jre* flag.

-jre zacPackage

Required if you use "O" as an option for the *-vm* flag. Specifies a ZAC-published JRE.

*-D*name=value

Specifies one or more Java system property to the Java VM when it is invoked by the bootstrapper.

—option

Pass the flag "-option" (with a single hyphen) to the Java application as it starts up.

*-ms*number

Sets the JVM initial heap size for your client application (in megabytes).

*-mx*number

Sets the JVM maximum heap size for your client application (in megabytes).

-nolaunch

Updates a ZAC package without launching it.

-noquery

Disables the dialog that prompts the user for an alternate installation directory when the bootstrapper is run. By default this is enabled.

-noprogress

Disables the display of the download progress meter.

-noupdate

Launches a ZAC application without attempting to update it. Requires a previously successful download.

-verbose

Enables verbosity for classloading in the Sun JVM. This can be useful for tracking down problems with missing class dependencies on the client.

-help

View a list of the available options. Each option also has a shorthand version; These are indicated in the output of the `-help` command.

3 Developing with WebLogic ZAC

This section describes the Application Programmatic Interface (API) for the Zero Administration Client, including the following topics:

- Introduction
- The WebLogic ZAC API
- Implementing with WebLogic ZAC

Introduction

This document describes how to use the ZAC API to ZAC-enable your Java applications. You should also be familiar writing a WebLogic client application, which introduces all of the services and facilities within the WebLogic environment.

WebLogic ZAC (Zero Administration Client), lets you automate the distribution and maintenance of your application software. ZAC removes the burden of manual software distribution, installation, re-installation, upgrades, bug-fix patches, and data distribution. It keeps your application software always up-to-date on your client machines via the Internet or intranet. ZAC's services can be made automatic and transparent to the end-user. Updates are fast and efficient, since ZAC only transmits the minimum changes required to bring each client up-to-date.

ZAC is an implementation of the W3C specification, the HTTP Distribution and Replication Protocol. In addition to a GUI wizard, the ZAC Publish Wizard, that makes it easy to publish software on a ZAC-enabled WebLogic Server. ZAC also has an API with which you can incorporate the same functionality directly into your Java applications. Note that ZAC is not supported using the 1.3.0 JDK with or without Java HotSpot TM.

Note: The JRE Update Frequency property set in the ZAC Publishing Wizard is not supported as of Version 4.5.1, and you need to use one of the following methods to update your JRE when necessary.

When a JRE is packaged with an application, it will be reinstalled on the client only if:

(a) The entire JRE installation directory is deleted from the client AND either the client executable OR the ZAC bootstrap routine is run; or

(b) A new version of the JRE library package is published on the WebLogic Server AND the bootstrap routine (not the client executable) is run from the client.

No package versioning takes place on the server (unless the version number is made part of the package name); it is only possible to "revert" a package if it has not already been published.

When to use the ZAC API

ZAC can be used as a wrapper around your existing Java applets and applications, or can be incorporated into your applications via the ZAC API. If you want to distribute and automatically update your Java software with ZAC, you do not really need to use the ZAC API; the ZAC Publish Wizard allows you to specify everything necessary to publish your application and make it available to client machines. For information on publishing applications with ZAC, see Publishing with WebLogic ZAC.

You should consider building ZAC into your Java code if you need to closely control the ZAC update services with your application. Your application can respond automatically when new updates are published, or offer more control to the user over when to accept new software updates. Another use of the ZAC API is to write applications that administer updates for other applications, or non-executable data on the client machine. For instance, maintaining a large dataset on each client machine,

such as a copy of a large corporate intranet that must be kept up-to-date on every Sales person's field laptop. Downloading the entire dataset every time a change is made might take a long time over the network, but when a ZAC-enabled application finds newly published versions, it downloads *only* the required changes.

How ZAC deploys applications

ZAC applications must be published on a ZAC-enabled WebLogic Server to be made available to WebLogic clients. You do not have to use the ZAC Java API in your application code in order to publish the application with ZAC; as part of the publishing process, ZAC supplies a tiny bootstrap executable that your client can download, and the bootstrapper will then handle initial download, subsequent updates (configurable), and starting of the published application. This bootstrap executable is compiled for each target platform's operating system, so that Java need not be pre-installed on the client.

You publish the bootstrap by posting a link for the executable from any web page; after the user downloads and runs the bootstrapper, an icon is installed on the desktop that the user can double-click to start the initial installation and subsequently to start the application itself. Before starting the published application, the bootstrapper checks for new updates on the server and upgrades the application as necessary. Subsequent updates will be more efficient since only the changes are downloaded.

You can exercise more control over when an application is updated by using the ZAC API to add code to your Java client. When the bootstrapper is generated by the ZAC Publish Wizard (see Using the Publish Wizard), you may uncheck the option so that it will *not* automatically update the client from a newly published ZAC package; rather, you can add the ZAC update functionality to the application itself, enabling the user can choose when to update from the application's user interface.

The WebLogic ZAC API

Package-weblogic.zac

```
Class java.lang.Object
  Class weblogic.zac.ZAC
```

```
    (implements weblogic.drp.events.ProgressListener,
    weblogic.drp.common.DRPConstants)
  Class weblogic.zac.ZACLog
```

Use the `ZAC` class to manage ZAC packages on the client machine from within your application. Most likely, you will use only a very small subset of the methods in this class, such as the `update()` method to update the ZAC package on the client machine from the publishing WebLogic Server.

The `ZACLog` class provides information about the published ZAC packages in use by your client, and the most recent updates for each of these packages.

Implementing with WebLogic ZAC

This section discusses the following topics:

- Importing Packages
- Updating ZAC Applications
- Using ZACLog to Query the Latest Updates
- Restarting a ZAC Client Application
- Using WebLogic Events with ZAC
- Packaging Libraries with Your ZAC Application

Importing Packages

To use the `ZAC` class in your Java application or applet you must import the WebLogic ZAC package, as well as the package that supports all WebLogic clients. For example:

```
import weblogic.zac.*;
import weblogic.common.*;
```

Updating ZAC Applications

Your application should create and use a ZAC object to reference and update a ZAC package. The ZAC constructor requires that you specify the following details about the ZAC package you want to administer:

- The address of the WebLogic Server where the ZAC package is published
- The name of the published ZAC package on the WebLogic Server
- The local installation directory of the ZAC package on the client machine

You can use the `ZACLog` class to obtain information about each ZAC package in use by your client application, and then pass that information to the ZAC constructor. The code below illustrates how you might do this:

```
// Find the address of each publishing server
// and connect to each in turn
ZACLog zl;
ZAC zac;
// Obtain an enumeration of ZACLog(s) for each published
// package this application is dependent upon
Enumeration enum = ZACLog.getUpdateLogs();
while (enum.hasMoreElements()) {
  zl = (ZACLog)enum.nextElement();
  // Construct a URL of the server address from each ZACLog
  t3url = "t3://" + zl.getZACHost() + ":" + zl.getZACPort() + '/';
  // Create a new ZAC object, and connect to the server.
  zac = new ZAC(zl.getZACHost(),
                zl.getZACPort(),
                zl.getZACName(),
                zl.getLocalDirectory());
  // Now perform the ZAC package update...
  zac.update();
}
```

In the above example, we use the `ZACLog.getUpdateLogs()` method to examine an enumeration of ZACLog instances. Each ZACLog instance refers to a separate ZAC package that this application depends upon. We use the properties of each ZACLog to create a ZAC object for each ZAC package. We then update each ZAC package using the `zac.update()` method.

If your application administers another ZAC package that it is not dependent upon, then the `ZACLog.getUpdateLogs()` method will be ineffective, since it only returns a ZACLog for each package that the current ZAC application uses. In this case, your application will need to pass explicit details about the publishing WebLogic Server and the name of the package to the constructor.

Once you have created a ZAC object, you may call its `update()` method. This checks for a newly published ZAC package, and updates the package on the client if necessary.

Using ZACLog to Query the Latest Updates

The details of each ZAC update are recorded into a set of ZACLog objects. Updates can occur when the application starts, or when the application initiates an update itself (as above). You obtain an Enumeration of ZACLog object from the latest update using the `ZACLog.getUpdateLogs()` static method. All previous ZACLog records are discarded; it is the responsibility of the application to maintain a record of updates if required.

The Enumeration returned by a call to the `getUpdateLogs()` method contains one ZACLog for each package that the application is dependent upon. You might process this Enumeration as shown in the following code:

```
  // Obtain Enumeration of ZACLog(s)
  Enumeration enum = ZACLog.getUpdateLogs();
  ZACLog zl;
  // Process each ZACLog...
  while(enum.hasMoreElements()) {
    zl = (ZACLog)enum.nextElement();
    // Print the ZAC package name
    System.out.println("ZAC log for package" + zl.getZACName());

    // Print the ZAC update status
    switch(zl.getUpdateStatus()) {
    case ZACLog.UPDATE_NONE:
      System.out.println("ZAC update status: No update was
necessary.");
      break;
    case ZACLog.UPDATE_FAILURE:
      System.out.println("ZAC update status: FAILED!");
     System.out.println("Details: " + zl.getUpdateFailureString());
      break;
```

```
    // This is where the real work starts
    case ZACLog.UPDATE_SUCCESS:
      System.out.println("ZAC update status: Completed
successfully");

      // Get info about the update
      int fileCnt = zl.getUpdateFileCount();
      long binarySize = zl.getUpdateByteCount();

      String details;

      if (fileCnt == 0) {
        details = "Update Success, 0 files updated, 0 bytes
transferred.";
      }
      else {
        if (fileCnt == 1) {
          details = "Update Success, 1 file updated, ";
        }
        else {
        details = "Update Success, " + fileCnt + " files updated, ";
        }

        if (binarySize > 1000) {
          details += (binarySize / 1000) + " KBytes transferred.";
        }
        else {
          details += (binarySize) + " bytes transferred.";
        }
      }
      System.out.println("Details: " + details);

      // Report which files were updated
      System.out.println("The following files were updated:");
      Enumeration fl = zl.getUpdateFileList();
      File zacroot = zl.getLocalDirectory();
      while (fl.hasMoreElements()) {
        String path = (String)fl.nextElement();
        File updated = new File(zacroot, path);
       System.out.println("Updated: " + updated.getAbsolutePath());
      }
    }
  }
```

The example above shows how to retrieve detailed information about the latest ZAC update from the ZACLog. First, the name of the ZAC package to which the ZACLog refers is obtained and printed out to the console. Next, the status of the ZAC update is queried and a `switch` statement is used to act upon the possible outcomes. The `getUpdateStatus()` method will return one of the following constants:

`ZACLog.UPDATE_NONE`
: No update was necessary; the package was up-to-date.

`ZACLog.UPDATE_FAILURE`
: The update failed in some way. More information can be obtained about the failure by calling the `getUpdateFailureString()` and the `getUpdateFailure()` methods. This returns a String containing a stack trace and the `Throwable` exception that occurred respectively. An update may fail due to a server error, a communications interruption, or a lack of disk space on the client machine.

`ZACLog.UPDATE_SUCCESS`
: Indicates that a successful update occurred. Several other pieces of information about the `ZACLog` are available when an update is successful, such as the size of the data transferred and the files that were updated.

The names of the updated files returned by the `getUpdateFileList()` method are relative to the installation directory into which the ZAC package was installed. In the above example, a FILE object is constructed from the relative pathname, appended to the path returned by the `getLocalDirectory()` method, giving the full path name of each file.

Changes that an update makes to an application will not be reflected until it is restarted.

Restarting a ZAC Client Application

If your client application was started using the ZAC bootstrapper, you may request it to be restarted by sending a `ZAC.ZAC_EXIT_RESTART` flag to the `System.exit()` method. Here is an example:

```
System.exit(ZAC.ZAC_EXIT_RESTART);
```

Since, the ZAC bootstrapper was used to start the application, it can also catch the exit status. When it receives the `ZAC.ZAC_EXIT_RESTART` status, it restarts the application. Prior to restarting, the bootstrapper will also update the application and other ZAC packages it is dependent upon if configured to do so. You usually configure this when you publish the application with the ZAC Publish Wizard.

Using WebLogic Events with ZAC

You can incorporate WebLogic Events into your client application to have it respond instantly when a new version of a ZAC package is published on a WebLogic Server.

Note: WebLogic Events are deprecated with the 6.0 release of WebLogic Server.

The WebLogic Server generates a new event for the `WEBLOGIC.ZAC.UPDATE.myPackage` topic when a new update of the package named *myPackage* is published on the WebLogic Server. When your application registers an interest in the event topic `WEBLOGIC.ZAC.UPDATE` and sets the `sink` flag to true, your application is notified when a new package version is published. For information about events, see Using WebLogic Events.

The following code shows how a client connects to the WebLogic Server and registers interest in the publication of a ZAC package. In this example, the client registers interest in the `"WEBLOGIC.ZAC.UPDATE.myPackage"` event topic.

```
ZACLog zl;
String t3url = null;
while (enum.hasMoreElements()) {
  ZACLog zl = (ZACLog)enum.nextElement();
  // Test for a match to the package name
  if (zl.getZACName().equals("myPackage")) {
    t3url = "t3://" + zl.getZACHost() +
            ":" + zl.getZACPort() + '/';
    break;
  }
}

if (t3url != null) {
  // Now connect to the server.
  T3ServicesDef t3services = getT3Services(t3url);

  // Register interest in the update event...
  // Create an Evaluate object to be used in the registration
  Evaluate eval =
```

```
      new Evaluate("weblogic.event.evaluators.EvaluateTrue");

    // Create an Action object parameter for the registration
    // To have the notification returned to the client, the client
    // must implement ActionDef and the Action must be instantiated
    // with the client object, i.e. "this".
    Action act = new Action(this);
    // Create the EventRegistrationDef object
    EventRegistrationDef erd =
      t3services.events()
        .getEventRegistration("WEBLOGIC.ZAC.UPDATE.myPackage",
                              eval, act, true, true, 1);

    // Finally, we register interest in the event
    int regid = erd.register();
  }
```

The client must implement the `action()` method, part of the ActionDef interface. In the `action()` method, you client may act upon notification of a published update, and update itself. Here is an example:

```
public synchronized void action(EventMessageDef ev) {
  System.out.println("Notification of an " +
                     ev.getTopic() +
                     " Event received.");
  zacUpdate = true;
  notifyAll();
  }
```

The `action()` method is usually synchronized to prevent multiple threads from executing in it simultaneously. In this implementation, we simply print a message to the console, set the private variable `zacUpdate` to `true`, and wake up other application threads to handle the event by calling the `notifyAll()` method. The `zacUpdate` variable indicates to the suspended application thread that is should perform a ZAC update.

Note: It is not good practice to perform time-consuming operations in a callback method such as this, since the event notification is called from another thread. Small operations that return quickly are acceptible, but in our case, we may decide to perform a ZAC update, which may take some time so is best handled from one of the client application threads.

Packaging Libraries with Your ZAC Application

Your published ZAC application should be published with any required libraries that it depends upon. The ZAC subset of the Weblogic classes is distributed with any ZAC application by default and is necessary for its operation, regardless of whether the application uses the ZAC API or not. These classes are downloaded to the client in the *zac.jar* file and placed in a *lib* directory below the ZAC application installation directory. For this reason, applications that use the ZAC API need not deploy the ZAC classes, since they are supplied by default.

However, applications that use WebLogic resources that are not included in the *zac.jar* file, or any other non-default libraries or packages, must be configured so that those resources are also installed on the client machine. This can be achieved in two ways, described below.

Including Libraries within a ZAC Package

If you include the required classes, jars, libraries, or any other data under your ZAC application's publish directory, they will automatically be deployed with that package and installed under the same relative package directory on the client machine.

If you application depends on classes outside of the application package, you must make sure that they are in the CLASSPATH of the ZAC application. You can specify a runtime CLASSPATH for your application in the ZAC Publish Wizard (see Using the Publish Wizard); the CLASSPATH must be specified relative to the root installation directory.

Making a ZAC Package Depend upon Another ZAC Package

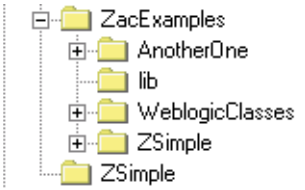
If you intend to publish several ZAC applications that use the same libraries or classes, you can save disc space on the client machine by publishing the shared components as a separate package. You then make each ZAC package dependent upon the shared package. The packages that your application depends upon, as well as your application itself, will be included each time your client application is updated.

When ZAC installs a package on a client, everything is installed under a root directory. The ZAC application you originally published will be installed in a subdirectory of the same name under this root directory. Any other packages will also be stored in subdirectories with the same name as the package under this root directory. ZAC only

has knowledge of the packages it has installed under the root directory. The location of the install-root directory is either defined when the package is published, or the package may be configured to allow the user to choose its location at install time.

In this example, the ZAC root directory is called ZacExamples, under which there are two packages, ZSimple and WeblogicClasses. In this case, the ZSimple package might depend upon WeblogicClasses.

For two ZAC packages to share a dependency on another package, both ZAC packages must be installed under the same root directory.



This illustration shows the directory structure when another package (called AnotherOne) has been installed under the same root directory. Both packages can share dependency on the same WeblogicClasses package, and both will update it if necessary.

Note: If not installed under the same root, each will maintain separate copies of the same package under their corresponding root directories. This is the default behavior; otherwise, a ZAC package might unintentionally update a package that another package was dependent upon.