Oracle Application Server 10.1.3 for JDeveloper Beginners

So you've built your first JDeveloper web application using the 1001 Oracle examples. You're boss is impressed. "Wow, that's one of the best web-pages I've ever seen Jones" (you're name is Jones isn't it?). And with a wave of the hand as your boss walks away, you hear the words "deploy it".

Welcome to the world of J2EE application servers and Oracle's latest flagship Oracle Application Server 10.1.3.

J2EE Application Servers

In order to run a JDeveloper web application or any Java based web application for that matter, it must be deployed to a J2EE compliant application server. To be J2EE compliant the vendor requests a license from Sun and tests their server technology against the Compatibility Test Suite. On passing the tests the vendor is granted the license to show the J2EE brand. From your point of view this is a good thing as it guarantees to a certain level that the J2EE application server supplies standard functionality.

There are a number of J2EE compliant server technologies available: Apache Tomcat, BEA WebLogic and JBoss to name a few. Oracle Containers for J2EE, or more commonly referred to as OC4J, is Oracle's own J2EE flavour. You already know OC4J if you've run a web application in JDeveloper as it contains an embedded OC4J. Also later Oracle Forms programmers will be familiar with OC4J as the Forms IDE does something very similar in running Web Forms. The obedient IDE JDeveloper has been invoking the embedded OC4J each time you run your application, deploying to the J2EE *container*, and calling your favourite web-browser to show the first web page of your application, all running off your local machine. The embedded OC4J runs within a Java Virtual Machine (JVM) from a standard Java Development Kit (JDK) utilising ADF Runtime Libraries and your application's source code.

But now it's time to play with the big boys. In uttering the words "deploy it" your boss has ensured that you can't keep playing with the local embedded OC4J team. You have to now deploy your application to an application server that multiple users can access.

As mentioned JDeveloper supports deployment to JBoss, Tomcat and WebLogic. Of course it also supports Oracle's own Oracle Application Server or OAS. OAS is made up of a number of separate processes designed to respond to requests such as returning a web page. One such process is in fact an OC4J container where you will deploy your applications to. Just like the embedded OC4J container within JDeveloper, the OAS equivalent runs within a JVM under a standard JDK and is an ideal location to deploy your applications.

OAS 10.1.3 Options

As usual any discussion on Oracle technology should mention specific versions. This article was written with the 10.1.3.0.0 OAS release in mind, namely OAS 10g Release 3. The 10.1.3.0.0 release only includes the J2EE option at this time, namely "Integrated Webserver, J2EE Server, Process Management" which fits nicely into the scope of this article. Those familiar with previous versions of OAS will note that options for Business Intelligence and other OAS smarts including support for deploying Oracle Forms and Reports are missing. These are

reported to be scheduled for a later 10.1.3 release of the OAS. Don't worry, I'm sure Oracle won't forget about Forms!

OAS Installation

Assuming your local friendly DBA gives you access, OAS provides the command line utility *opmnctl*. The Oracle Process Manager and Notification Server (OPMN) is responsible for monitoring the OAS components. *opmnctl* is the command line utility for interacting with OPMN. From experience OAS administrators at some point breakdown to using the command line tools over Oracle Enterprise Manager (OEM) due to the inherit bugs in the OEM interface. While this may not be true of later versions of the OAS OEM, particularly since the 9.0.4 series, it's the same argument of why good DBAs use SQL*Plus and not OEM and products like Quest's Toad. Choose your own poison as such.

opmnctl on OAS can be used to monitor the status of your OAS components. Listing 1 shows how to do this.

```
C:\product\10.1.3\OracleAS_1\opmn\bin>opmnctl status -l

Processes in Instance: keen.acute
-------------+-------------+------+--------+------------+---------+---------+------
ias-component | process-type |  pid | status |        uid | memused |  uptime | ports
-------------+-------------+------+--------+------------+---------+---------+------
OC4J          | home         | 3184 | Alive  | 1626213932 |   59844 | 0:16:43 | jms:12601,ajp:12501,rmi:12401
HTTP_Server   | HTTP_Server  | 1908 | Alive  | 1626213931 |   53972 | 0:16:21 | https1:443,http2:7200,http1:80
ASG           | ASG          |  N/A | Down   |        N/A |     N/A |     N/A | N/A
```

Listing 1

Using the *opmnctl status -l* options reveals the components currently running, their process ID, status, user ID, amount of memory consumed, uptime and the ports of the various services within the components. This provides a handy summary for checking what's running, process IDs if you later need to kill a wayward process, and which port numbers to connect to.

In addition the *opmnctl* options *startall* and *stopall* are used to start and stop the application server as a whole.

You'll note that 3 components are running. OC4J is the J2EE container you will deploy to and we'll talk more about this soon. ASG refers to OracleAS Guard, designed for disaster recovery on a standby OAS site and beyond the scope of this paper.

The Oracle HTTP_Server (OHS) acts as a front-end listener to all web requests. It is responsible for interrogating the requests and passing them to the other OAS components such as the OC4J to render a response. As it is the first port of call to the applications the OHS can also configure load balancing (as complimentary to the features provided by Web Cache if installed at your site).

From the output of the *opmnctl status -l* command you can see that the OHS is listening on 3 ports using 2 protocols (namely http and https). Assuming the OC4J component is up you can enter the URL *http://<hostname>:<portnumber>* to access the OAS splash-screen where the <*hostname*> is the OAS server name and <*portnumber*> is the *http1* port number listed from the *opmnctl* output (from Listing 1 - 80).

To log into the OAS OEM from the splash-screen, from the right side panelBox select the option "log on to Oracle Enterprise Manager 10g Application Server Control", then supply the OAS oc4jadmin account credentials. Once logged in to the OAS OEM, it will reveal something similar to Figure 1. This shows the OC4J container named *home* within an application server *keen* on a localhost *acute*.

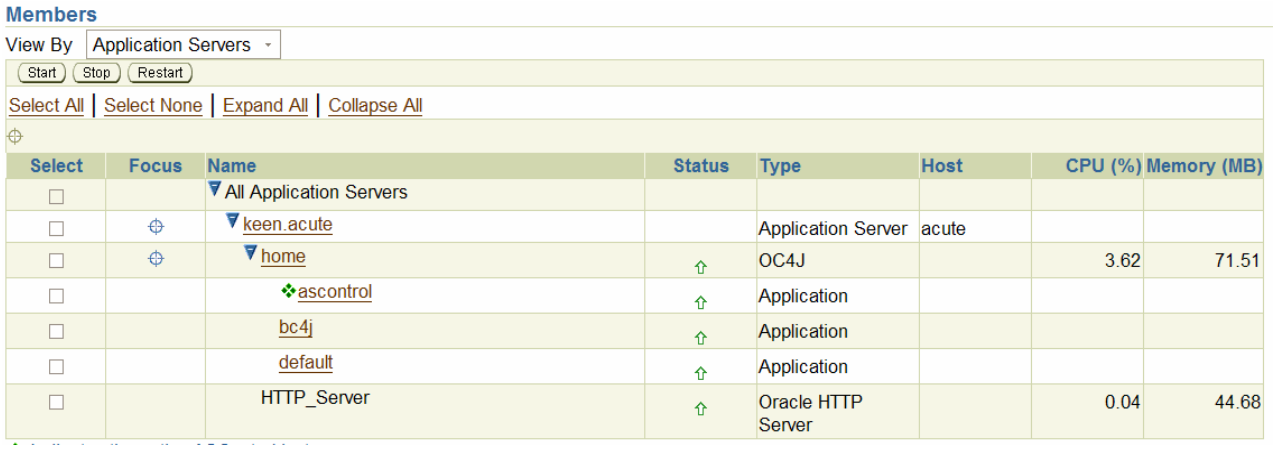


Figure 1

By default the 10.1.3 OAS *home* OC4J contains 3 preconfigured applications. *ascontrol* represents the OAS Application Server Control and is the actual J2EE OEM application. A sweet point for vendors is when they can write their own custom applications that work within their server products. In this case OEM has been written with the latest Java web technologies and deployed to OAS for you to use.

I have yet to find a satisfactory answer what the *bc4j* node represents.

The 3rd *home* OC4J component is *default*. On deploying a J2EE application to this OC4J instance, the *default* application serves as the parent J2EE container. At the *default* level you can set a number of configurations and install Java libraries which are automatically inherited by the child applications unless specifically overridden.

The *HTTP_Server* represents the same OHS you discovered on issuing the *opmnctl status -l* command.

Configuring JDeveloper for deployment

In order to deploy your app from JDeveloper, you need to setup an application server connection, a deployment descriptor file for your project as well as deploy the ADF Runtime Libraries to the application server. The JDeveloper 10.1.3 documentation has an excellent section entitled "Deploying Applications" which will show you step by step how to do this.

The ADF Runtime Libraries contain the code for ADF Business Components, TopLink, the ADF Model/Binding Layer, ADF Faces and other proprietary Oracle technologies.

To install the ADF Runtime Libraries using the JDeveloper utility you need to shutdown the OAS, then run the JDeveloper utility on the same machine. Alternatively the utility can run across the network providing you have created a network mount point on the server. As a last

resort if either of these options aren't available to your environment the libraries can be manually copied to your OAS.

The utility backs up files under the OAS *bc4j* directory (*bc4j* is the old name for ADF) under a new directory BC4JAR<version>, and copies in the new files. You'll note that the *bc4j* directory is at the root level of the OAS. More on this later.

JDeveloper licensing issues

As a side note, JDeveloper licensing is required before deploying your JDeveloper application. Recently Oracle announced that they have made JDeveloper 10.1.3 free to use, and they donated ADF Faces to the Apache MyFaces project under the Open-Source Apache 2.0 Licence. This may lead you to believe you can freely use JDeveloper in a production environment. However it's a little more complicated than that.

Strictly speaking the JDeveloper 10.1.3 IDE is free to use. The JDeveloper ADF libraries are also free to use in a production environment with one caveat: the ADF libraries are installed on a licensed 10g Oracle Application Server 10g. If you choose to deploy to an alternative J2EE server technology such as JBoss, a per-processor or per-seat license applies.

The caveat for ADF Faces is a little murkier as the Apache donation was only recently announced. It may no longer apply to the Apache donated code but it's currently unclear if the ADF Faces version included within the ADF libraries still has this restriction. More information about the JDeveloper licensing issues can be found at:

http://www.oracle.com/technology/products
/jdev/htdocs/jdevpricefaq.html
http://www.oracle.com/technology/products
/jdev/htdocs/faq_adffaces_apache.html

Deployment Options

Once you've created the deployment descriptor for your application, created a connection to OAS in JDeveloper, and installed the ADF Runtime Libraries to the application server, you are ready to deploy your application. JDeveloper provides multiple methods for deploying. The choice you make depends on which part of the development process you're at, meaning development, testing or production, as well as what you're more familiar with. This becomes more obvious once we explain the options.

In JDeveloper if you right click on the deployment descriptor, the context menu will show among other options the following three:

- Deploy to → <list of application servers>
- Deploy to WAR
- Deploy to EAR

The first option is the easiest and deploys straight to one of the preconfigured application server connections you created earlier. In essence this option is the same as the Deploy to EAR option removing the manual task of installing the EAR file on your application server.

The 2nd and 3rd options are similar in that either the WAR or EAR file is generated to your local file-system, which you must then manually copy to the application server and install from there. While these latter options requires more work they do provide more control.

As a rule the first option is easiest when in the development phase as generally your OAS administrator doesn't care how you deploy. However in most organisations deploying your application to a test or production environment requires a formal change control process. As such the 2nd and 3rd options are typically used, attaching the EAR or WAR files to a change-control request.

In addition the 2nd and 3rd options are preferred by traditional Java web programmers due to the added control. This raises the question how do they manually deploy the application to OAS? 2 methods: either selecting the OC4J container in OEM and then the Deploy button under the Applications tab. You'll note you're also given buttons to Undeploy, Redploy and Restart each application as demonstrated in Figure 2.

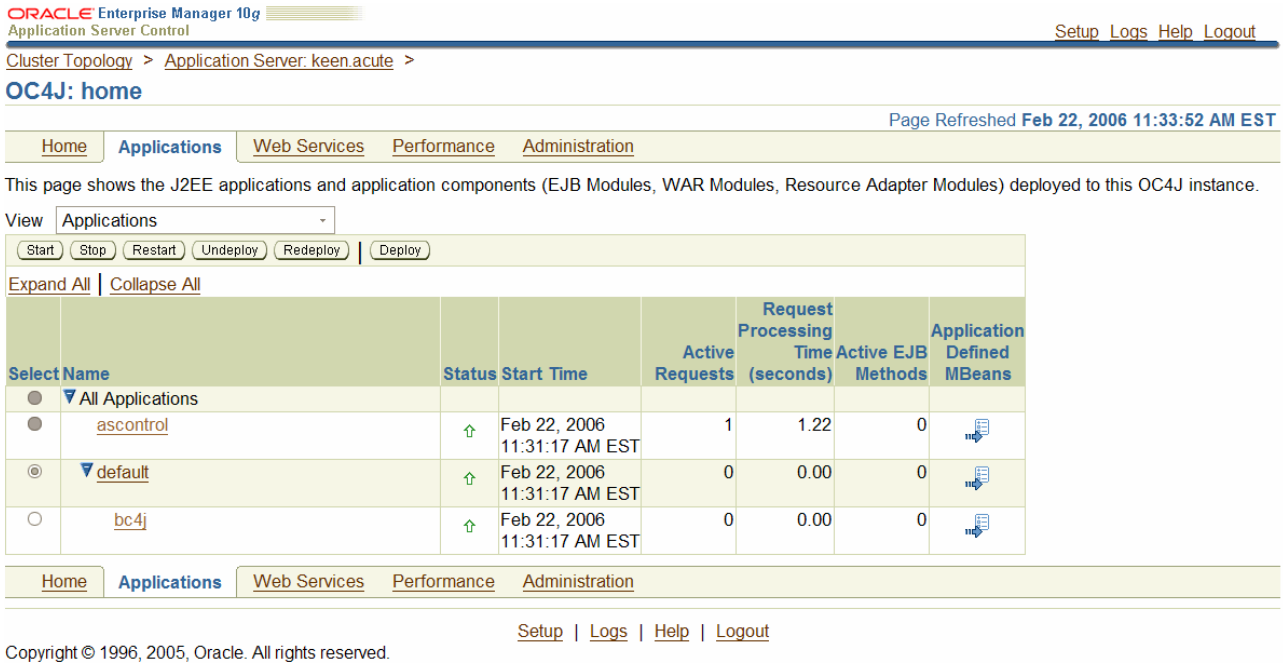
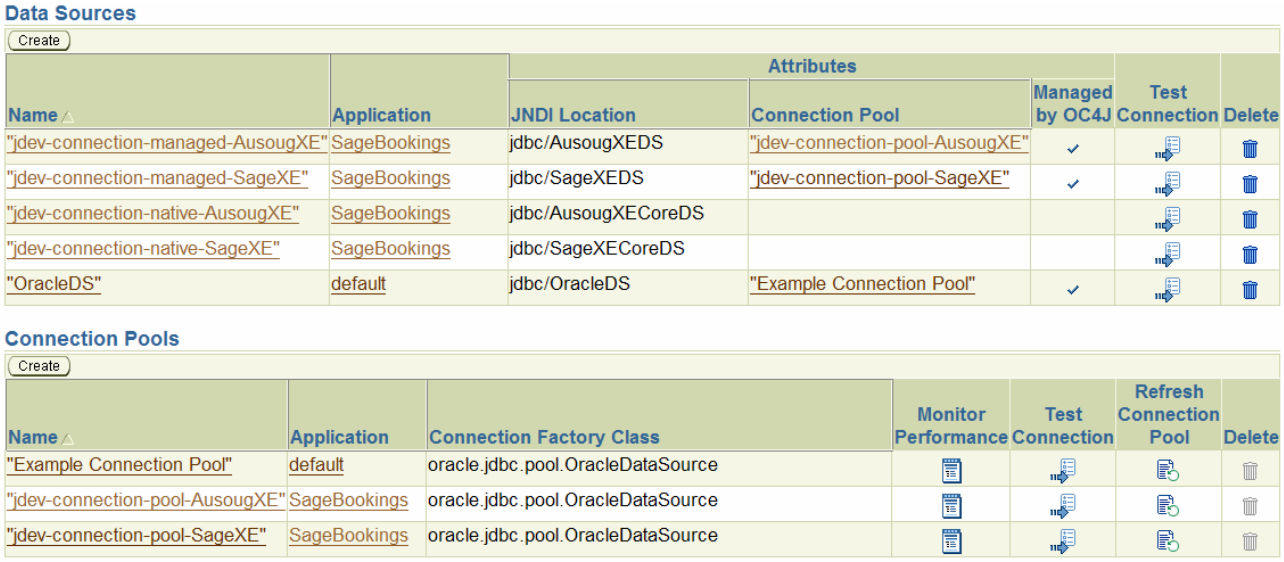


Figure 2

The 2nd method is via a Java command line utility supplied with OAS named *admin_client.jar*. The OAS Oracle Containers for J2EE Configuration and Administration Guide has good documentation on how to use this utility.

One caveat to be aware of is if you select either the Deploy to <application server> or Deploy to EAR option, then among other files, JDeveloper includes the file data-sources.xml. This file contains a list of all your JDeveloper connections (where the passwords are encrypted). Once deployed if you click on your OC4J home's Administration tab, then the JDBC Resources Go-To-Task option under the Services section, you'll see a list of the Data Sources and Connection Pools that have been created on your behalf as shown in Figure 3.



Data Sources

Create

Name	Application	Attributes		Managed by OC4J	Test Connection	Delete
		JNDI Location	Connection Pool			
"jdev-connection-managed-AusougXE"	SageBookings	jdbc/AusougXEDS	"jdev-connection-pool-AusougXE"	✓		
"jdev-connection-managed-SageXE"	SageBookings	jdbc/SageXEDS	"jdev-connection-pool-SageXE"	✓		
"jdev-connection-native-AusougXE"	SageBookings	jdbc/AusougXECoreDS				
"jdev-connection-native-SageXE"	SageBookings	jdbc/SageXECoreDS				
"OracleDS"	default	jdbc/OracleDS	"Example Connection Pool"	✓		

Connection Pools

Create

Name	Application	Connection Factory Class	Monitor Performance	Test Connection	Refresh Connection Pool	Delete
"Example Connection Pool"	default	oracle.jdbc.pool.OracleDataSource				
"jdev-connection-pool-AusougXE"	SageBookings	oracle.jdbc.pool.OracleDataSource				
"jdev-connection-pool-SageXE"	SageBookings	oracle.jdbc.pool.OracleDataSource				

Figure 3

A JDBC Data Source is a named abstraction of a specified connection and its credentials and can be used by multiple applications rather than hard-coding the credentials.

Note that for each JDeveloper database connection, 2 Data Sources are created in OAS with the names *jdbc/<connectionname>DS* and *jdbc/<connectionname>CoreDS.* The later uses a native JDBC driver for connecting to the database. The former is a OC4J managed wrapper to the native JDBC driver and makes use of OAS C*onnection Pools*.

Conversely if you selected the Deploy to WAR option, the data-sources.xml file is not created and deployed. If your application uses a JDBC Data Source you would then need to manually create it within OAS.

JDBC URL vs JDBC Data Sources

A consideration and one that isn't immediately obvious is the use of a JDBC URL or JDBC Data Source for your ADF BC application. These 2 options are responsible for your application connecting to your database.

On creating an Application Module within your new ADF BC application, the JDeveloper wizards allow you to select a database connection you have previously created. This configuration is accessible from the right-mouse-menu *Configurations* option, then selecting the *Edit* button for the default *AppModuleLocal* configuration as shown in Figure 4.

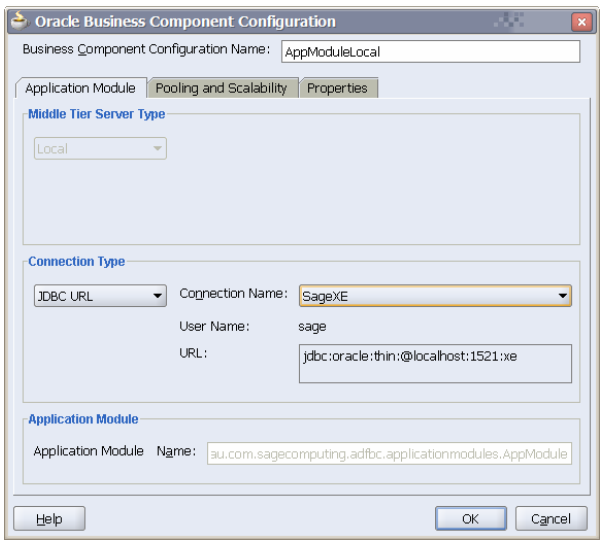


Figure 4

The JDBC URL option is the default and the JDBC connection credentials are "borrowed" from the database connection you first selected in the wizard. Within your application these settings are stored with the bc4j.xcfg.

Once your application is deployed to OAS it uses this hardcoded JDBC URL embedded in the bc4j.xcfg file. The obvious disadvantage of this method is the hard-coding and the OAS administrator is unable to change the connection credentials without hacking the bc4j.xcfg file. Note however if you wish to use a dynamic JDBC URL during the life of your application this is the option for your application (a dynamic JDBC URL is one where you change the connection credentials at runtime programmatically).

The JDBC Data Source Connection Type option on the other hand allows you to make use of the named OAS Data Sources. This assumes you've either created them previously in OAS, or alternatively deployed the application using the EAR option which included the data-sources.xml file as previously discussed.

The format of the JDBC Data Source is *jdbc/<connectionname>DS* if you wish to use the OC4J managed JDBC driver, or *jdbc/<connectionname>CoreDS* to use the native JDBC driver. The main advantage for the OAS administrator on using this option is the administrator can change the credentials via OEM. The second advantage is multiple applications using the same named Data Source can share the same connection pool and the efficiency gains of this mechanism.

ADF Runtime Libraries, JDKs and OC4J Instances

OAS 10.1.3 is certified to run against J2SE JDK 1.4.2 and 5.0, of which 5.0 is installed by default. JDeveloper 10.1.3 and the supplied ADF Runtime Libraries are also shipped with JDK 5.0. Any application you develop under JDeveloper 10.1.3 therefore has a natural dependency on the specific version of the ADF Runtime Libraries which in turn have a dependency on JDK 5.0.

Given your new found skills in JDeveloper and your boss's enthusiasm you're likely to write and deploy more applications. However eventually Oracle is going to release a new version of

JDeveloper with a new set of ADF Runtime Libraries, or you may also find a bug in your application that requires a new JDK version. This creates a configuration issue on your OAS as you may have several JDeveloper applications running fine under the old ADF Runtimes Libraries or JDK, and a fixed application that requires later versions. What do you do rather than upgrading all the old applications and undertake an expensive regression testing exercise (you wouldn't take the somewhat riskier approach and skip it of course!)?

Within OAS you are able to create additional OC4J instances beyond the default home OC4J. These prove an ideal location to install your applications rather than the default home. The command line *createinstance* utility provided with OAS 10.1.3 allows you to create additional OC4J containers. The OAS Oracle Containers for J2EE Configuration and Administration Guide has more information on how to use this utility. Note that in the 10.1.3 release of OAS, unlike previous versions, OEM does not yet provide the ability to create additional OC4J instances. It is rumoured this feature will be available in 10.1.3.1.

To override the JDK version for a specified OC4J instance, you make changes to the opmn.xml file for that instance, specifically to the <data id="java-bin"> tag within the <ias-component id="oc4jname"> tag, where oc4jname is the new OC4J instance.

In browsing your Oracle OAS directories, note under the *j2ee* directory there is a *home* subdirectory. This is your original OC4J instance's file system. Also notice the new OC4J instance you just created is also located under the *j2ee* root. When you deploy your applications from JDeveloper, this is where the files go.

Prior to OAS 10.1.3, loading alternative ADF Runtime Libraries or any additional 3rd-party libraries for that matter relied on you manually copying the libraries into the new OC4J instance's *applib* directories. Unfortunately the JDeveloper ADF Runtime Installer isn't sophisticated enough to be able to deploy to separate OC4J instances on the OAS. This left you with the task of manually copying the files into the new OC4J container. By default any application deployed into a separate OC4J instance will search the parent OC4J's *applib* directories before defaulting to the *home* OC4J's libraries.

OAS 10.1.3 introduces a new feature known as the Class Loader. Unlike the previous version, in addition to searching the library hierarchy as explained previously, OAS allows you to upload separate versioned libraries that can be mapped against a particular application, and can be shared by multiple applications. The advantage over the previous method is you now explicitly state the library dependencies for an application rather than hoping the administrator has dropped the correct versioned libraries in the correct OC4J directories. More than ample documentation on the Class Loader is supplied in the Oracle Containers for J2EE Developer's Guide.

Once you decide to deploy to this new OC4J instance and you use JDeveloper to do it, ensure within the application server connection you change the specified OC4J instance, or alternatively create a new connection to deploy to the separate instance.

Conclusion

This article was designed to open beginner JDeveloper's to the concepts of deployment, JDBC URLs and JDBC Data Sources, and OAS configuration issues for future expansion. From experience the author has found that OAS administration falls into the hands of over-worked DBAs and as such uninformed decisions can be made on the installation of J2EE applications.

This can have unhealthy affects on systems in the future, including user support and the success of the project as a whole.

While administration of J2EE servers may fall into the hands of 3rd parties, the JDeveloper must have an appreciation of end-to-end issues in order to be successful at the task. The boss's flippant call to "deploy it" opens a new array of issues that must be navigated in order to satisfy such a simple request.