ORACLE®

ATG WEB COMMERCE

Version 10.0.2

Installation and Configuration Guide

Oracle ATG
One Main Street
Cambridge, MA 02142
USA

ATG Installation and Configuration Guide

Document Version

Doc10.0.2 INSTALLATGv1 4/15/2011

Copyright

For information about Oracle’s commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/us/corporate/accessibility/index.html.

Oracle customers have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/support/contact.html or visit http://www.oracle.com/accessibility/support.html if you are hearing impaired

Contents

1 Installing the ATG Platform

This document describes how to install and configure ATG on the JBoss, WebSphere, or WebLogic application servers. This chapter covers the following topics:

Document Conventions

This guide uses the following conventions:

- `<ATG10dir>` represents the ATG installation directory (`C:\ATG\ATG10.0.1`, for example)
- `<JBdir>` represents the Red Hat JBoss home directory (`C:\jboss\jboss-eap-5.1\jboss-as`, for example)
- `<WLdir>` represents the Oracle WebLogic home directory
- `<WASdir>` represents the IBM WebSphere home directory

Default Ports

This guide uses the *`hostname:port`* convention in URLs. The default HTTP ports for the application servers are:

- JBoss: 8080
- WebLogic: admin server 7001

- WebSphere: 9080

Important Terms

This section defines terms used throughout this guide.

ATG products. Umbrella name for the software suite, particularly the platform.

ATG installation. Collective name for the tools, files, classes, etc. used for developing and assembling J2EE applications.

ATG application. A piece of software installed independent of the platform, which can be included as a module or set of modules in a Nucleus-based application.

ATG server. A configuration layer that is available to be added to other configuration layers by the application assembler when assembling an EAR.

Dynamo Administration UI. Web pages used to configure and monitor the ATG installation.

Component. A Java object instance of a specific configuration for a JavaBean that is registered with Nucleus.

Nucleus-based application. An assembled EAR file created out of components managed by ATG's Nucleus component manager, running on the application server.

Product Requirements

You must install your application server before you install the ATG platform. See your application server documentation for installation information.

Before you run the ATG setup program, make sure you have a supported JRE in place on your system, and that the JVM is in your system PATH. (**Note:** The stand-alone ATG Control Center includes its own JRE.)

You should enable GZIP compression for static files. See your application server documentation for information.

For a detailed list of system requirements for the ATG platform, see the Supported Environments page (http://www.atg.com/en/products/requirements/).

JBoss-Specific Requirements

After installing JBoss, modify the JVM arguments. Go to `<JBdir>/bin/run.conf|bat` and edit the `JAVA_OPTS` line. ATG suggests the following settings:

```
JAVA_OPTS="-server -Xms2048m -Xmx3072m -XX:MaxPermSize=768m
-XX:MaxNewSize=768m -Dsun.rmi.dgc.server.gcInterval=3600000 –
Dsun.rmi.client.gcInterval=3600000"
```

If you are setting up a JBoss instance that will be dedicated to lock management, you can run that instance with a smaller heap size, since the lock manager does not serve pages. To do this, ATG recommends creating a new `run.bat|sh` file referring to a new `run.conf` file.

Duplicate the `run.bat|sh` and `run.conf` files and rename the duplicates (for example, `runLockMan.sh` and `runLockMan.conf`). In the `runLockMan.bat|sh` file, change the following section to point to the new configuration file:

```
# Read an optional running configuration file
if [ "x$RUN_CONF" = "x" ]; then
    RUN_CONF="$DIRNAME/LMrun.conf"
fi
if [ -r "$RUN_CONF" ]; then
    . "$RUN_CONF"
fi
```

The `runLockMan.conf` file should include the following settings:

```
JAVA_OPTS="-server –Xms512m –Xmx512m -XX:MaxPermSize=128m
-XX:MaxNewSize=128m
-Dsun.rmi.dgc.server.gcInterval=3600000"
```

Using ACC Scenarios in JBoss

In order to create scenarios in the ATG Control Center (ACC), you must add the following three JAR files to your classpath:

```
<JBdir>/common/lib/jboss-javaee.jar
<JBdir>/common/lib/jsp-api.jar
<JBdir>/common/lib/servlet-api.jar
```

To do this, copy the files into the `/lib` directory of your standalone ACC installation, then modify the `bin/startClient.bat` file to include the three JARs in the class path.

Disabling Session ID Checking in JBoss

If you are using JBoss on UNIX and expect to run multiple ATG servers within a single JBoss instance (as may be the case during development or demonstrations), edit the JBoss `run.conf` script by adding the following line to the end of the file:

```
JAVA_OPTS="${JAVA_OPTS} -Dorg.apache.catalina.connector.Request
.SESSION_ID_CHECK=false"
```

This allows your browser to use a single `jsessionid` cookie for both instances, avoiding unnecessary errors.

If you are running ATG on Windows, session ID checking is disabled by default, using the –`disableSessionIdCheck` flag in the `startDynamoOnJBOSS.bat` script (see the Using the startDynamoOnJBOSS Script section for additional flags).

WebLogic-Specific Requirements

If you are using WebLogic and want to run the ACC in a dedicated VM (see Starting the ACC in a Dedicated VM in this guide), you must add the following tag to the `config.xml` file inside the `<security-configuration>` tag:

```
<enforce-valid-basic-auth-credentials>
  false
</enforce-valid-basic-auth-credentials>
```

See your WebLogic documentation for information on the `config.xml` file.

To use XA data sources with WebLogic, add the following line to your `<ATG10dir>/home/servers/servername/localconfig/GLOBAL.properties` file:

```
localTransactionModeInitialization=false
```

In order to create scenarios in the ACC, you must add the `<WLdir>/server/lib/wlclient.jar` file to your class path. To do this, copy `wlclient.jar` into the `/lib` directory of your standalone ACC installation, then modify the `bin/startClient.bat` file to include `wlclient.jar` in the class path.

If you are planning to run SQLJMSAdmin on your WebLogic installation, you must change the `session-timeout` value in the SQLJMSAdmin `webModule\WEB-INF\web.xml` file from zero to a positive number. ATG recommends setting the timeout value to 30.

Controlling Page Recompilation on WebLogic

When you run ATG applications on WebLogic, WebLogic's JSP container manages JSP compilation. If you are running WebLogic in development mode, modified pages are automatically recompiled when they are requested, ensuring that the `.java` files associated with the pages are up to date. To prevent performance degradation due to unnecessary page recompilation, when you run WebLogic 10 in production mode, page recompilation is automatically disabled (`.jsp` files should not change on a production environment, so in theory recompilation will never happen; but disabling recompilation ensures that it will not be triggered by a timestamp change).

Although recent WebLogic versions automatically disable page recompilation in production mode, you may want to manually disable recompilation if you are in a testing phase, but not yet running in production mode. Unnecessary recompilation may distort performance tests and slow down your quality assurance process.

To disable page recompilation, create a `weblogic.xml` file (or modify an existing one) in the `WEB-INF` directory of each web application you want to include in your EAR file. In the `weblogic.xml` file, set these two parameters to -1:

- `pageCheckSeconds` specifies the interval in seconds between stale checks for an individual JSP. When a request for a JSP is received, if the last stale check on this page was longer ago than the number of seconds that `pageCheckSeconds` is set to, a new stale check is performed, and if the page is determined to be stale, it is recompiled. The default in development mode is 1 second. Setting this parameter to -1 disables stale checking.

- `servet-reload-check-secs` specifies the interval in seconds between checks of a web application's `WEB-INF/classes` directory to see if any servlets have been recompiled (and therefore need to be reloaded). The default in development mode is 1 second. Setting this parameter to -1 disables checking.

The following example illustrates disabling both of these checks in the `weblogic.xml` file:

```
<weblogic-web-app>
  <container-descriptor>
    <servlet-reload-check-secs>-1</servlet-reload-check-secs>
  </container-descriptor>
  <jsp-descriptor>
    <page-check-seconds>-1</page-check-seconds>
  </jsp-descriptor>
</weblogic-web-app>
```

WebSphere-Specific Requirements

The information in the following sections applies only to those using the WebSphere Application Server.

If you have installed the WebSphere Network Deployment version, when you run the ATG installer you must select the *IBM Websphere - cluster setup* option (even if you are not actually using clustering).

Running WebSphere on AIX

If using WebSphere on AIX, to avoid errors when importing application data using ATG import scripts, you must set the following in the `<ATG10dir>/home/localconfig/postEnvironment.sh` file:

```
JAVA_ARGS="${JAVA_ARGS} -Djava.net.preferIPv4Stack=true"
```

You must also set it in the WebSphere environment:

1. In WebSphere Admin, go to Servers > Application servers > server >Java and Process Management > Process Definition > Java Virtual Machine.
2. Under Generic JVM arguments set the following:

   ```
   -Djava.net.preferIPv4Stack=true
   ```

If you do encounter this problem, you will see errors such as the following:

```
Error: Jan 30, 2008 12:45:01 PM javax.jmdns.JmDNScloseMulticastSocket
WARNING: closeMulticastSocket() Close socketexception
java.net.SocketException: The socket name is not available on this system.
```

XA Data Sources on WebSphere

To use XA data sources with WebSphere, add the following line to your `<ATG10dir>/home/servers/servername/localconfig/GLOBAL.properties` file:

```
localTransactionModeInitialization=false
```

Creating ACC Scenarios on WebSphere

In order to create scenarios in the ACC, you must add the `<WSdir>/AppServer/j2ee.jar` file to your class path. To do this, copy `j2ee.jar` into the `/lib` directory of your standalone ACC installation, then modify the `bin/startClient.bat` file to include `j2ee.jar` in the class path.

Using ATG Multisite on WebSphere

If you are using ATG's multisite feature on WebSphere, in order to use virtual context roots, do the following:

1. In WebSphere Admin, go to Servers > Server Types > WebSphere application servers > *`server_name`* > Web Container settings > Web Container > Custom Properties.
2. Set the com.ibm.ws.webcontainer.invokefilterscompatibility property to true.

In order for session recovery to function across a multisite installation, make sure to set the following properties as indicated on each application server:

- Enable URL rewriting – Enabled
- Enable protocol switch rewriting – Enabled
- HttpSessionReuse – True

Sun T1000 and T2000 Requirements

By default the Sun T1000 and T2000 systems run a server that uses port 9010. ATG's lock management components also use this port. If you are using lock management, you must either disable the server or change your lock manager to use a different port.

To disable the server:

1. Log in as root.
2. Enter the following command:

   ```
   mv /etc/rc2.d/S95IIim /etc/rc2.d/K95IIim
   ```

3. Stop the service:

   ```
   /etc/rc2.d/S95IIim stop
   ```

To change ATG lock manager port assignments, when you configure your lock management components, use the following settings:

1. For the `ClientLockManager` port assignment in `<ATG10dir>/home/localconfig/atg/dynamo/service /ClientLockManager.properties:`

   ```
   useLockServer=true
   lockServerPort=39010
   ```

2. For the `ServerLockManager` port assignment in `<ATG10dir>/home/servers/`*`servername`*`/localconfig/atg/dynamo/service/ServerLockManager.properties`:

   ```
   port=39010
   ```

See the *Locked Caching* section of the *ATG Repository Guide* for information on configuring lock managers.

Running the ATG Setup Program

The ATG platform installer is available as a self-extracting Windows executable (`ATG10.0.1.exe`) or UNIX binary file (`ATG10.0.1.bin`). This distribution file includes the following products:

- ATG Adaptive Scenario Engine
- ATG Business Commerce
- ATG Consumer Commerce
- ATG Portal
- ATG Content Administration

Follow these steps to install the platform:

1. Run the `ATG10.0.1.exe` or `ATG10.0.1.bin` file to start the setup program.

   Note: If you are installing on a Linux variety that includes GCJ, in order to avoid installation errors you must specify a JVM that includes the `javax.swing` classes, which are not included in GCJ. Use the following command:

   ```
   $sh ./install.bin LAX_VM path_to_java_executable
   ```

   For example:

   ```
   $sh ./ATG10.0.1_678.bin LAX_VM /usr/local/j2sdk1_4_2_03/bin/java
   ```

2. After you accept the terms of the license agreement, select the installation folder for the ATG software (`C:\ATG\ATG10.0.1` or `/home/ATG/ATG10.0.1`, for example).
3. Select the ATG products you want to install.
4. Select your application server.
5. If installing for **JBoss**, enter the following configuration information :
   - the RMI port your Nucleus-based applications will use (defaults to 8860)
   - the listen port that JBoss uses to listen for incoming connections (defaults to 8080)
   - the JBoss home directory (`C:\jboss-eap-5.1\jboss-as`, for example)
   - the JDK home directory (`C:\j2sdk1.6.0_22`, for example)

   If installing for **WebLogic**, enter the following:

- the RMI port your ATG applications will use (defaults to 8860)
- the listen port that WebLogic uses to listen for incoming connections (defaults to 7001)
- the WebLogic home directory
- the path to your WebLogic domain directory (`C:\oracle\user_projects\domains\mydomain`, for example)
- the JDK home directory (`C:\j2sdk1.6.0_22`, for example)

If installing for **WebSphere**, enter the following:

- the RMI port your ATG applications will use (defaults to 8860)
- the port that WebSphere uses to listen for incoming connections (defaults to 9080)
- the WebSphere home directory (`C:\WebSphere\AppServer`, for example)
- the name of the WebSphere server (server1, for example)
- the node on which the WebSphere server is installed (Typically, the node name is the same as the host machine name.)

JBoss Installation Results

The ATG installer creates a JBoss server named `\atg` in your `<JBdir>\servers` directory. This server is based on the `\default` JBoss server without modifications. See your JBoss documentation for information on JBoss servers.

Note: Do not deploy multiple ATG application EAR files to a single JBoss server.

WebLogic Installation Results

The ATG setup program adds a `protocol.jar` file to the WebLogic domain directory you specified during the installation process. Before you start WebLogic, open the `<WLdir>\user_projects\domains\`*`your_domain`*`\startWebLogic.{cmd|sh}` file and add the `protocol.jar` path to the beginning of the `CLASSPATH` variable. For example:

```
set CLASSPATH=C:\WebLogic\user_projects\domains\mydomain\protocol.jar;
%WEBLOGIC_CLASSPATH%;%POINTBASE_CLASSPATH%;%JAVA_HOME%\jre\lib\rt.jar;
%WL_HOME%\server\lib\webservices.jar;%CLASSPATH%
```

Note that you should **not** move this file into the `lib` directory for your domain, nor should you include it in the `lib` for your Nucleus-based application. It should be in the `CLASSPATH` for the application server.

WebSphere Installation Results

If you are not using CIM to configure your installation, you must manually register the ATG URL providers, following this procedure:

1. Copy `protocol.jar` from the `<ATG10dir>\DAS\lib` directory to the `\lib` directory of your WebSphere installation.

2. Register the following URL providers in the WebSphere Admin Console (see your WebSphere documentation), using the specified settings:

```
name = dynamosystemresource
streamHandlerClassName =
atg.net.www.protocol.dynamosystemresource.Handler
protocol = dynamosystemresource

name = appmoduleresource
streamHandlerClassName = atg.net.www.protocol.appmoduleresource.Handler
protocol = appmoduleresource
```

Performing a Maintenance Installation

If you have any of the ATG platform products installed and would like to install additional platform products, rerun the `ATG` setup program. The maintenance installer lists the products that have not been installed yet, allowing you to select the ones you want. If you need to reinstall any of the ATG platform products that are currently installed on your system, you must uninstall the ATG platform completely (see Removing the ATG Platform from Your System) and run the setup program again.

Note: If you have installed any ATG patches, you must uninstall them before running the maintenance installer. Once the maintenance install is complete, reinstall the patches. See the `PatchReadme` files under `<ATG10dir>/patch` for instructions.

Installing the ATG Control Center on a Client Machine

This section explains how to install a standalone version of the ACC (ACC) on a client machine, when you do not need a full ATG installation. It covers the following topics:

- Downloading the ACC Installer
- Installing the ACC on a Windows Client
- Installing the ACC on a UNIX Client

Note: To use the standalone version of the ACC, the client machine must have the J2SDK installed.

Downloading the ACC Installer

Contact your ATG sales representative to obtain one of the following ACC distribution files:

- `ACC10.0.1.exe` (Windows)
- `ACC10.0.1.jar` (UNIX)

Note: You cannot use any other version of the ACC with ATG 10.

Installing the ACC on a Windows Client

To install the ACC on a Windows client:

1. Run the `ACC10.0.1.exe` file to start the setup program.
2. After you accept the terms of the license agreement, select the destination folder for the ACC. The default is `C:\ATG\ACC10.0.1`. Click **Browse** to specify a different directory.
3. Enter a name for the ACC program folder on the Windows Start menu.
4. The installer displays the settings you selected. Review the setup information and click **Next** to start the installation, or **Back** to change any of the settings.

Installing the ACC on a UNIX Client

To install the ACC on a UNIX client:

1. Change the permissions on the downloaded installer so you can execute it.
2. Run the binary:

   ```
   ./ACC10.0.1.bin
   ```

3. Accept the license agreement.
4. Provide an install directory.

When finished, exit the installer.

Installing ATG Development Tools for Eclipse

ATG offers a set of development tools for the open source Eclipse Platform (http://www.eclipse.org). For the latest information about the ATG Eclipse plug-ins, point your Eclipse Update Manager to `http://www.atg.com/eclipse`.

Use the Eclipse Update Manager to install the ATG Eclipse plug-in:

1. Open the Eclipse Workbench and select **Help** > **Software Updates** > **Find and Install**.
2. In the Feature Updates dialog, select **Search for New Features to Install** and click **Next**.
3. Check the **Web Tools Platform (WTP) Updates** option and click **Finish**.
4. Select a mirror site if prompted and click **OK**. The Update Manager searches for features.
5. In the Search Results window, expand the list and select the **Web Standard Tools (WST)** and **J2EE Standard Tools (JST)** projects. Install these projects in your Eclipse installation.
6. In the Eclipse Workbench, select **Help** > **Software Updates** > **Find and Install** again.

7. In the Feature Updates dialog, select **Search for New Features to Install** and click **Next**.
8. In the Install dialog box, click **New Remote Site**.
9. In the New Site Bookmark dialog box, enter ATG in the Name field and `http://www.atg.com/eclipse` in the URL field. Click **OK.** The Update Manager adds an ATG bookmark to the Feature Updates view.
10. Check the ATG bookmark and click **Finish**. The update manager searches for ATG tools you do not have installed.
11. In the Search Results window, expand the ATG bookmark and select the plug-ins you want to install, then click **Next**.
12. Accept the license agreement, then click **Next**.
13. Click **Finish** to install the plugins. You will have to restart Eclipse before using your newly installed plugins.

To learn more about using the ATG Eclipse plugins, see the ATG documentation under Help > Help Contents in Eclipse after you have installed them.

Using the Configuration and Installation Manager (CIM)

ATG's Configuration and Installation Manger (CIM) cuts down on the complexity of configuring multiple ATG applications. A series of text-based wizards guide you through configuration procedures, ensuring that necessary steps are completed, and that steps are performed in the correct order.

CIM is a text-based, menu-driven interface that walks you through several configuration tasks that are commonly susceptible to error. Menus are dynamically generated based on your selections to provide choices appropriate for your installation.

The installation guides for individual products contain specific information on what CIM accomplishes for those products, but in general, CIM handles the following configuration areas:

- Datasource configuration
- Database table creation and data import
- ATG server instance creation and configuration (see Creating Additional ATG Server Instances)
- Application assembly and deployment

The result is a functional installation that can be used as a starting point for further configuration. CIM does not replace configuration steps that require a running ATG application, or for which a suitable user interface exists.

Note: CIM does not configure a scenario or process editor server. See the *ATG Multiple Application Integration Guide* for information on scenario editor servers.

To use CIM, do the following:

1. Install your application server and database software.

   WebSphere Note: In order to use CIM to configure an ATG installation for WebSphere, your WebSphere installation needs to use cell deployment. There is a separate WebSphere installer for this type of installation, denoted by ND. Also, note that if you run WebSphere as a root user, CIM must also be run as a root user.

   WebLogic Note: If you are using offline deployment in WebLogic, you can deploy only a single ATG EAR file per domain. CIM deploys the EAR file to WebLogic's autodeploy directory, and that EAR is loaded by any running server instance.

2. Download and install your ATG products.

   Note: In order to use CIM, you must install ATG Content Administration.

3. Navigate to:

   ```
   <ATG10Dir>\home\bin
   ```

4. Enter the command:

   ```
   cim.bat|sh
   ```

5. Follow the prompts to configure your installation. To access the online help, enter H at any point.

Removing the ATG Platform from Your System

Use the following methods to remove the ATG platform from your system.

On Windows: Use the Add/Remove Programs function in the Windows Control Panel.

On UNIX: Go to the `<ATG10dir>/uninstall/.ASE10.0.1_uninstall` directory and run `Uninstall_ATG_10.0.1`.

2 Running Nucleus-Based Applications

Nucleus-based applications are assembled into EAR files that include both the application and ATG platform resources, and which are then deployed to your application server. The ATG platform installation includes the modules required to create `QuincyFunds.ear`, a sample J2EE application that includes the Quincy Funds demo and the Dynamo Administration UI. The Quincy Funds demo requires the SOLID SQL database, which is included in the ATG distribution for evaluation purposes.

Once the ATG installation is complete, you can assemble, deploy, and run the `QuincyFunds.ear` application. You can then access the Quincy Funds demo and the Dynamo Administration UI through your web browser, and connect to the application with the ACC.

This chapter covers the following topics:

Starting the SOLID SQL Database

Before you start up a Nucleus-based application, make sure the SQL database you intend to use is running. For evaluation purposes, `QuincyFunds.ear` is preconfigured to work with the SOLID SQL database included in the ATG software distribution. This database comes fully configured with data for all of the ATG demo applications, including the Quincy Funds demo.

JBoss Note: JBoss by default assumes XA drivers, which some ATG applications use; however, there are no XA drivers for SOLID. To enable multiple non-XA resources in JBoss, add the property in bold text to the `jbossjta-properties.xml` file, under the `<property depends="arjuna" name="jta">` tag:

```
<property depends="arjuna" name="jta">
  <property name="com.arjuna.ats.jta.allowMultipleLastResources"
  value="true"/>
```

You may still see warnings in your log file, but ATG applications will run correctly. To suppress these warnings, add the following to your `jboss-log4j.xml` file:

```
<category name="com.arjuna.atg.jta.logging">
  <priority value="ERROR"/>
</category>
```

Note: The SOLID database is not supported on AIX. (See the Configuring Databases and Database Access chapter for information about configuring ATG products to work with other databases.)

To start SOLID:

On **Windows**:
On the Start go to **ATG 10.0.1** > **Tools** > **Start SOLID Server**.

On **UNIX**:
Run the `<ATG10dir>/home/bin/startSolid` script.

Note: On UNIX, SOLID looks for the `libpam.so` and `libpam.so.1` files. If you are running Solaris, you may need to create symbolic links to the following files before running the `startSolid` script. To create symbolic links, do the following:

1. Make sure you are logged in as the root user.
2. Type the following commands:

```
ln -s /usr/lib/libpam.so /usr/lib/libauth.so
ln -s /usr/lib/libpam.so.1 /usr/lib/libauth.so.1
```

3. Log out and log in again under your own user name.

Note: By default, SOLID starts in the background. On UNIX, you can run the SOLID server in the foreground, to see any SOLID error messages that occur. To start SOLID in the foreground, switch to the `<ATG10dir>/home/` directory and type `bin/startSolid -f`.

Running the Demos and Reference Applications

You can use the `runAssembler` utility (see the *ATG Programming Guide*) to create EAR files that contain the ATG reference applications. Include the following modules for each application:

Demo	Required Modules
Quincy Funds	`DSSJ2EEDemo`
Motorprise	`MotorpriseJSP`

Once the SOLID server is running, you can start up your application server. If you chose to install it, the `QuincyFunds.ear` application has already been deployed by the ATG installer. (For information about starting up, see your application server documentation).

Note: Because of the way the demo databases are configured for evaluation purposes, you cannot include more than one demo module when assembling your application.

Note: If you are running JBoss on Windows, you can access the demo from your Windows Start button. If you selected the default shortcut location, go to Programs > ATG 10.0.1 > ATG Adaptive Scenario Engine > Quincy Funds Financial Services.

On WebSphere, before using a demo, set the following properties in the `/atg/dynamo/servlet/pipeline/DynamoHandler.properties` file:

```
fixRequestURI=true
fixServletPath=true
```

The following table lists the default URLs for accessing the demos on the supported application servers.

Demo	URL and Documentation Link
Quincy Funds (Personalization)	`http://`*`hostname`*`:`*`port`*`/QuincyFunds` ATG Quincy Funds Demo Documentation
Motorprise (B2B Commerce)	`http://`*`hostname`*`:`*`port`*`/Motorprise` ATG Business Commerce Reference Application Guide

Starting the SQL-JMS Admin Interface

The ATG platform includes a browser-based administration interface for its SQL JMS message system. This interface makes it easy to view, add, and delete SQL JMS clients, queues, and topics. To use the SQL-JMS Admin interface, include the `SQLJMSAdmin` module in your application.

To access the interface, point your browser to the following URL:

```
http://hostname:port/sqlJmsAdmin
```

To learn more about the SQL JMS system, see the *ATG Programming Guide*.

Starting ATG Web Services

The ATG platform includes a number of preconfigured web services that provide remote access to ATG repositories and various personalization and commerce features. (For detailed information about these services, see the *ATG Repository Guide*, *ATG Personalization Programming Guide*, and *ATG Commerce Programming Guide*.) These services are packaged in three separate applications:

```
<ATG10dir>/DAS/WebServices/repositoryWebServices.ear
<ATG10dir>/DPS/WebServices/userprofilingWebServices.ear
<ATG10dir>/DCS/WebServices/commerceWebServices.ear
```

You can include any of these web services in an assembled EAR file by including the module that contains the desired services. For example, to include the Commerce services, specify the `DCS.WebServices` module when you invoke the `runAssembler` command (see the *Assembling Applications* section of the *ATG Programming Guide* for information on using `runAssembler`).

Connecting to the Dynamo Administration UI

The Dynamo Administration UI gives you quick access to the following features:

Configuration Manager
Modify configuration for ATG server instances.

Component Browser
Browse the Nucleus component hierarchy.

Admin ACC
Start up the ACC.

Change Password
Change administrator passwords.

JDBC Browser
Browse a database through a JDBC connection, examine database metadata, create and drop tables, and execute database queries.

Performance Monitor
View performance statistics on ATG applications.

Web Service Administration
Create and manage web services.

Batch Compiler
Precompile JHTML pages to prevent any delay the first time they load.

Configuration Reporter
Display reports about ATG component properties and environment.

Personalization Administration
Find, edit, and create user profiles. If you have access to the Business Control Center, that should be used instead.

You can access the Dynamo Administration UI at `http://hostname:port/dyn/admin`. (On Windows, you can also select the **ATG Dynamo Server Admin** icon in the Tools folder of the ATG 10.0.1 program group.) The initial user name and password are:

User Name: **admin**
Password: **admin**

For information about including the Dynamo Administration UI when you assemble an EAR file, see the *Including the Dynamo Administration UI* section of the *Developing and Assembling Nucleus-Based Applications* chapter of the *ATG Programming Guide*.

Connecting to the ATG Business Control Center

If your application includes the `BIZUI` module, you can use the ATG Business Control Center to create, preview, approve, deploy, and revise site content, as well as to access other ATG applications. To access the ATG Business Control Center, point your browser to the following URL:

```
http://hostname:port/atg/bcc
```

To learn more about the ATG Business Control Center, see the *ATG Content Administration Guide for Business Users*.

Starting the ATG Control Center

You can start the ACC in several ways, depending on whether you're starting it locally in relation to your Nucleus-based application, or on a separate client.

Note: If you are using a UNIX variant, the shell from which you start the ACC must support X11 forwarding. Depending on your client, you may need to install X11 packages, or use Xming or equivalent tools.

Note: Due to a Java bug, if you are running Java 6, you cannot run the ACC in the same virtual machine as the application server. You can run the ACC in a dedicated VM, or install the following IBM iFix:

```
http://www-01.ibm.com/support/docview.wss?uid=swg24027328
```

To connect to a Nucleus-based application from a client machine, you must use the client version of the ACC (see Installing the ATG Control Center on a Client Machine in the Installing the ATG Platform chapter for more information). Note that for a Nucleus-based application to accept connections from the ACC, all of the following must be true:

- The application includes the `DAS-UI` module.
- The `rmiEnabled` property of the `/atg/dynamo/Configuration` component is set to `true`.

- The adminPort property of the /atg/dynamo/Configuration component is set to the listen port of your application server (for example, the JBoss default is 8080).

These settings are all part of the default configuration created by the ATG installer, so you generally do not need to configure them.

In addition, to enable the client version of the ACC to connect to an application, the application must include the DafEar.Admin module. This module is not included by default, so you must explicitly specify it when you assemble the application. See *Including the Dynamo Administration UI* in the *ATG Programming Guide* for more information.

JBoss Note: In order to connect to your running ATG application from any remote location (that is, not using localhost), you must start your JBoss server using the –b option. For example, on Windows use the following command:

```
run.bat -b 0.0.0.0
```

See your JBoss documentation for information on this and other settings.

Starting the ACC on a Server

If you're starting the ACC on the machine that's running your application server, you can run the ACC either in a dedicated VM or in the same VM as the application server.

Note: Starting the ACC in a dedicated VM requires more memory than starting the ACC in the same VM as the application server. Running the ACC and the application server simultaneously on a production server is not recommended, as it could affect performance.

Starting the ACC in a Dedicated VM

To start the ACC in a dedicated VM:

On **Windows**:
On the Start menu, click the **Start ATG Control Center** icon in the Tools folder of the ATG 10.0.1 program group.

On **UNIX**:
Go to <ATG10dir>/home/bin and type the command startACC.

You can also start the ACC in a dedicated VM through the Dynamo Administration UI:

1. Open the Dynamo Administration UI (http://*hostname*:*port*/dyn/admin, by default), and click the **Admin ACC** link.

   The Start ACC page appears, indicating the server VM on which the ACC will be started and the machine on which the ACC will be displayed.

2. Click the **Start ACC in Separate VM** button.

When the ACC starts up, it displays the Connect to Server screen. Enter a valid user name, password, and the RMI port number, and select the locale from the drop-down menu. By default, the initial settings are:

User Name: **admin**
Password: **admin**
Locale: **English (United States)**
Port: **8860**

Note that the host name appears as `localhost`. This value is not editable. To start up the ACC on a remote client machine, see Starting the ACC on a Client.

Starting the ACC in the Same VM as the Application Server

To start the ACC in the same VM as your application server, use the Dynamo Administration UI:

1. Open the Dynamo Administration UI (`http://`*`hostname:port`*`/dyn/admin`, by default) and click the **Admin ACC** link.

   The Start ACC page appears, indicating the server VM on which the ACC will be started and the machine on which the ACC will be displayed.

2. Click the **Start ACC in Server VM** button.

When the ACC starts up, it displays the Connect to Server screen. Enter a valid user name and password. By default, the initial settings are:

User Name: **admin**
Password: **admin**

Note that you cannot specify the host name, locale, or RMI port. The ACC automatically uses the values set in the Nucleus-based application.

Exporting RMI Objects

If the ACC displays an error message while trying to connect to the server, you may need to modify the arguments passed to the Java Virtual Machine by configuring Java Remote Method Invocation (RMI) to export RMI objects on a particular IP address. This can happen under either of the following conditions:

- The server or the client is running on a machine with multiple host addresses; or
- ATG is running on a machine that has a primary IP address other than `localhost`, but the IP address is not functional because the machine is offline.

If ATG is running on a multihomed server, you can enable RMI to export objects to a particular address by including the following switch in the `JAVA_ARGS` environment variable:

```
-Djava.rmi.server.hostname=IP_Address
```

For the IP address, specify the IP address or name of the host that the client uses to connect to the server. Alternatively, you can specify the name of the server instead:

```
-Djava.rmi.server.hostname=hostname
```

If ATG is running on a machine whose IP address is not functional because the machine is offline, use the following switch:

```
-Djava.rmi.server.hostname=localhost
```

Troubleshooting

If you encounter any errors while using the ACC, check the `<ATG10dir>/home/data/acc.log` file for information.

Starting the ACC on a Client

To start the ACC on a client machine and connect to an ATG application running on a remote application server:

On **Windows**:
Click the **Start ATG Control Center** icon in the ATG Control Center 10.0.1 program group on the Start menu.

On **UNIX**:
Go to the ACC 10.0.1 installation directory and run `bin/startClient`.

When the ACC starts up, it displays the Connect to Server screen. Enter a valid user name and password, and the RMI port number, and select the locale from the drop-down menu. By default, the initial settings are:

User Name: **admin**
Password: **admin**
Locale: **English (United States)**
Port: **8860**

In addition, you must specify the name of the host machine on which the Nucleus-based application is running. This is the name used to identify the machine on a network.

Logging in to a Different Nucleus-Based Application

When the client ACC connects to a Nucleus-based application, it compiles information about the modules in that application (see the *Working with Application Modules* chapter in the *ATG Programming Guide* for information). To disconnect the ACC from one application and connect to an application that includes a different combination of modules, close down the ACC and restart it to ensure that the ACC compiles all the necessary information.

Troubleshooting

If you encounter any errors while using the client ACC, check the `/data/acc.log` file in the ACC installation for information.

Stopping an ATG Application

How you stop an ATG application depends on your application server.

Stopping ATG Applications on JBoss

To stop an application, you can remove it from the `deploy` directory or shut down the application server. To shut down the server, go to `<JBdir>` and enter the following command:

Windows:

```
bin\shutdown -s hostname
```

UNIX:

```
bin/shutdown.sh -s hostname
```

On Windows, you can also use CTRL+C to shut down the JBoss server.

Stopping ATG Applications on WebLogic

You can stop an ATG application through the WebLogic Server Console. You do not need to shut down the application server to stop the application.

Stopping ATG Applications on WebSphere

You can stop an ATG application through the WebSphere administrative console. You do not need to shut down the application server to stop the application.

Using the startDynamoOnJBOSS Script

The `startDynamoOnJBOSS` script makes it easy to run ATG applications on the JBoss application server. It combines the manual steps of assembling the EAR file, copying over the SOLID data source definitions required by the demo applications, and starting the JBoss server, into one step. See the *Assembling Applications* section of the *ATG Programming Guide* for information on these manual steps and on `runAssembler` options.

Note: If you are using CIM to configure and deploy your EAR files, do **not** use `startDynamoOnJBoss` to start your servers. If you want to use the Windows shortcuts installed with ATG, edit the scripts to remove that step.

This script is intended to be used by developers who need to rapidly and iteratively build, deploy, and run applications to see their changes. You should *not* use this script on a production site, or under any circumstances where it is important to closely watch the deployment process, such as when deploying to multiple servers. Instead, deploy your application manually.

If you already have JBOSS_HOME set in your `dasEnv` file (this is set by the installer if you installed ATG for use with JBoss), then go to `<ATG10>\home\bin` and type:

```
startDynamoOnJBOSS –m module-list
```

Note: If you are including the Dynamo Administration UI in this EAR file, the `DafEar.Admin` module must precede any custom modules in the module list.

The script performs the following actions:

1. Calls `runAssembler` with a destination EAR file called `ATG.ear` in the `<JBdir>/server/atg/deploy` directory. This directory is created by the ATG installer.
2. Includes the modules you specify and their dependent modules in the EAR file. If none are specified, the script includes `DSS`, `DAS-UI` and all of their dependent modules.
3. Assembles the EAR in exploded format in development mode (rather than packed format or standalone mode).
4. Copies data source XML files into the `<JBdir>/server/atg/deploy` directory to ensure that there are valid data sources for standard logins such as admin, motorprise, publishing, svcss, and svcshared (see the Configuring Data Sources for JBoss section of this guide).
5. Calls the JBoss `run.sh/bat` script.

ATG starts up on JBoss with the specified modules.

The following table describes the syntax to use if you do not want the default settings to apply:

Intended Result	**Syntax**
You don't have a `JBOSS_HOME` environment variable set	You can specify a `JBOSS_HOME` when running the `startDynamoOnJBOSS` script: `bin\startDynamoOnJBOSS -j c:\jboss`
You want to use a specific ATG server	`bin\startDynamoOnJBOSS [servername]`
You want to use more modules than just DSS (the default)	`bin\startDynamoOnJBOSS -m` *`module-list`*
You want to use a JBoss server other than the `\atg` server created by the ATG installer	You can pass in a different server name using the `-c` flag: `bin\startDynamoOnJBOSS -c someOtherServer`
You want to use a different name for the EAR file	Pass in the new EAR file name using the `-ear` flag: `bin\startDynamoOnJBOSS -ear ServiceKnowledge.ear`

Intended Result	Syntax

You want to pass additional flags to `runAssembler`.	Pass in additional flags using the `–f` flag:
	`startDynamoOnJBOSS -f -run-in-place`
	Note that for flags that pass in additional information, those additional arguments have to go before the flag itself. For example, if you want to pass in the `runAssembler –prependJars` flag, use the following syntax:
	`-f C:\\myclasses.jar -f -prependJars`
	To see all syntax options for the `startDynamoOnJBOSS` script, run the script with the `-help` flag. Also see the *ATG Programming Guide*.
	Note that if you are using `runAssembler` alone, the `–layer` switch must precede the `–m` switch, however, this it may come after `–m` when using `startDynamoOnJBOSS`.

3 Configuring Nucleus Components

This chapter explains how to configure Nucleus components in your ATG installation. Components represent a particular configuration for a class. Many different components can be based on a single class, each representing a different set of properties for that class. When the class is instantiated from the component, it uses the component properties to configure it.

You can configure components in the following ways:

- Using the ACC
- Manually editing properties files
- Using the Dynamo Configuration Manager in the Dynamo Administration UI (changes are limited)
- Using the Component Browser in the Dynamo Administration UI (live components only, changes do not persist beyond restart)

This chapter covers the following topics:

Most of the information in this chapter applies only for applications running in development mode (see the *Developing and Assembling Nucleus-Based Applications* chapter of the *ATG Programming Guide* for the differences between development and standalone modes).

Working with Configuration Layers

Before changing the configuration of Nucleus-based applications, you should be familiar with the concept of *configuration layers*. This section covers the following topics:

- Understanding Properties Files
- Understanding Configuration Layers
- Accessing Configuration Layers in the ACC
- Global Configuration Changes
- Locking Configuration Layers

Understanding Properties Files

ATG application modules use properties files to configure Nucleus components. The base properties files are normally stored in the `config` subdirectory of the module, either as individual plain text files or as part of a JAR file (see Modifying Custom Module Resource Settings to configure alternative configuration paths). For example, much of the default configuration is determined by properties files stored in `<ATG10dir>/DAS/config/config.jar`.

Note: Do not modify the properties files in these JAR files to change configuration settings, or your changes will be overwritten when you install a new ATG platform distribution.

To see the properties files in your ATG installation, do the following:

1. Start the ACC.
2. Select **Pages and Components** > **Components by Path** from the navigation menu.
3. Open the `/atg/dynamo/Configuration` component. When the ACC Component Editor opens, click the **Configuration** tab.

Note that there are several `Configuration.properties` files. You can view the contents of these properties files by double-clicking the file names.

Understanding Configuration Layers

ATG platform configuration layers allow you to make configuration changes and preserve them locally, without modifying the base configuration. Layers contain properties files, and can be stacked in a variety of ways to create different configurations for different purposes. The configuration stack is determined from the `MANIFEST.MF` files for the ATG application modules included in the application.

Nucleus locates configuration properties by examining the properties files in the directories and JAR files specified by the configuration path or paths (a module can have any number of configuration paths). The paths for all modules used in your application are aggregated and ordered based on the module dependencies. The result is a combination of the property values found in each of the files or directories in the configuration paths. If the same property value is defined in more than one properties file, values found later in the configuration path (as determined by the module dependencies) override the values found earlier. The `localconfig` directory usually appears last in the configuration path, so that any properties defined there override default system settings.

For example, suppose you change the port number for ATG's internal RMI server, by setting the `rmiPort` property of the `/atg/dynamo/Configuration` component, and save the new value in the `localconfig` directory. The next time you start the application, Nucleus will take the value of the `rmiPort` property from `localconfig`, because it is the last directory in your configuration path.

Any changes you make to `localconfig` are preserved when you install a new ATG version.

For more information on modules, configuration layers, and properties files, see the *Nucleus: Organizing JavaBean Components* and the *Working with Application Modules* chapters of the *ATG Programming Guide*.

Accessing Configuration Layers in the ACC

When you modify a component's properties in the ACC, the updated properties file is stored in one of the following locations:

- The ACC's default configuration directory, initially set to `<ATG10dir>/home/localconfig`
- A server-specific directory if the component already has a configuration in that layer. For example, if you run an application that does not use the default ATG server, and you modify a component using the ACC, the updated properties file is stored in the `localconfig` directory for the ATG server used by that application.

Note: The ACC shows only the configuration layers used by the application to which you are currently connected.

Resetting the Default Configuration Layer

Unless you specify otherwise, the ACC editor saves all updates to a component's configuration in the default configuration layer. Components that you create, duplicate, or paste are also placed there.

The installation initially sets the default configuration layer to `<ATG10dir>/home/localconfig`. You might want to change the default configuration directory if you have multiple servers running different applications. For example, you might have one server running a customer service application and another running an online store.

You can set any unlocked configuration layer as the default.

You can change the default configuration layer on the server, so it affects all server clients and persists across all editing sessions; or only on the local client. If you change the default layer locally, the setting remains in effect until you shut down the host.

1. Navigate to the configuration layer that is currently set as the default, and open its `CONFIG.properties` file, or create one if it does not yet exist.

   For example, the ATG installation initially sets the default configuration layer to `<ATG10dir>/home/localconfig/`. Therefore, open this file:

   `<ATG10dir>/home/localconfig/CONFIG.properties`

2. Set `defaultForUpdates` to false.
3. Navigate to the desired configuration directory and open its `CONFIG.properties` file.

For example, to set `<ATG10dir>/home/servers/myNewServer` as the default configuration directory, open this file:

```
<ATG10dir>/home/servers/moogus/CONFIG.properties
```

4. Set `defaultForUpdates` to true.

To temporarily reset the default configuration layer within the ACC:

1. In the ACC, select **Set Update Layer** from the **Tools** menu.
2. When the **Set a Default Configuration Layer** dialog opens, select the configuration layer that you want to open by default.

Changing a Component in a Non-Default Configuration Layer

To change a component in a non-default configuration layer:

1. Select the component to edit,
2. Choose **File** > **Open Component in Layer**.

   The dialog box Select a Configuration Layer opens, listing the name and path of each configuration layer. Check marks identify the layers currently in use.
3. Select the layer to open and click **OK**. The component opens in a separate Component Editor window.

Global Configuration Changes

Global configuration settings are configured in the `GLOBAL.properties` (located in `config/config.jar`) file. The settings in this file control logging and log listeners and apply to all components in the `config` tree except those that set these properties explicitly themselves. To change these values, you must edit this file manually (see Changing Component Properties Manually later in this chapter), or override them by adding your own `GLOBAL.properties` file in another configuration layer.

Locking Configuration Layers

Locked configuration layers such as Dynamo Base are marked with a padlock icon. Properties in a locked layer cannot be edited. To lock a configuration layer, modify the `CONFIG.properties` file for that layer as follows:

1. Open the `CONFIG.properties` file for the layer to lock.
2. Add the following line to `CONFIG.properties`:

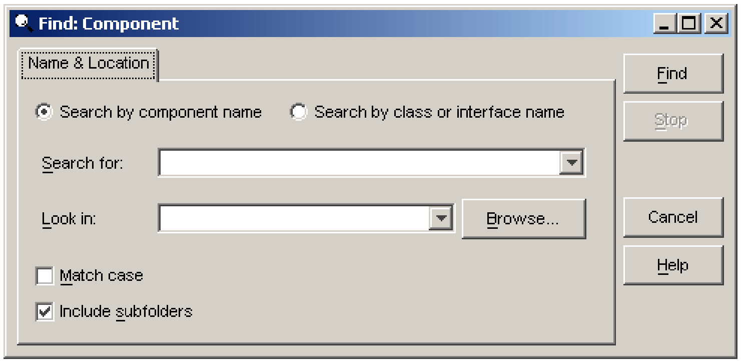
```
readOnly=true
```

Finding Components in the ACC

When changing ATG component configuration, you can use the ACC to search for components by name, class or interface.

To search for a component:

1. Choose **File** > **Find Component** in the main ACC window. The Find Component dialog box opens, as shown below.



● Search by component name ○ Search by class or interface name

Search for:

Look in:

□ Match case

☑ Include subfolders

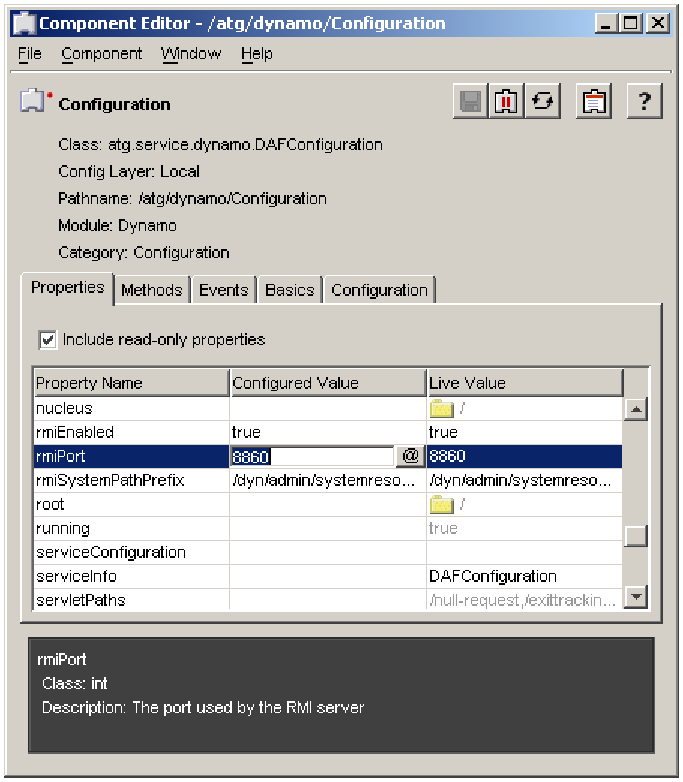
2. Click the radio button that indicates the way you want to search: **Search by component name** or **Search by class or interface name**.
3. Type the component name in the **Search for** field. You can search for partial names by using the asterisk (*) or question mark (?) wildcard symbols. If you want your search to be case-sensitive, check the **Match case** box.
4. Type the location you want to search in the **Look in** field or click the **Browse** button to select a directory from the component hierarchy. To search all folders within this directory, make sure the **Include subfolders** box is checked.
5. Click the **Find** button. The search results appear at the bottom of the Find Component dialog box.

Changing Component Properties with the ACC

The ACC provides a simple way to change many configuration settings. This section uses an example in which you change the port number of ATG's internal RMI server.

To change the port number:

1. Start the ACC.
2. Select **Pages and Components** > **Components by Path** from the navigation menu.
3. Open the `/atg/dynamo/Configuration` component.
4. When the ACC Component Editor opens, click the **Properties** tab. Scroll down to the `rmiPort` property:



Configuration

Class: atg.service.dynamo.DAFConfiguration
Config Layer: Local
Pathname: /atg/dynamo/Configuration
Module: Dynamo
Category: Configuration

☑ Include read-only properties

Property Name	Configured Value	Live Value
nucleus		/
rmiEnabled	true	true
rmiPort	8860	8860
rmiSystemPathPrefix	/dyn/admin/systemreso...	/dyn/admin/systemreso...
root		/
running		true
serviceConfiguration		
serviceInfo		DAFConfiguration
servletPaths		/null-request,/exittrackin...

rmiPort
Class: int
Description: The port used by the RMI server

Note: Certain expert-level properties are visible only if you select the **Show expert-level information** check box in the **Preferences** > **Tools** > Edit Preferences dialog box.

If the component has been started (indicated by a red dot), the Properties tab displays two columns of property values: the *Configured Value* and the *Live Value,* described in the table below. You can edit the value of any non-shaded property by clicking in its value cell and entering a new value.

Configured Value	Live Value
The value specified by the component's properties file	The current value, which may be different from the configured value
Changes to the value appear in the ACC immediately, but the changed values are not used to configure the component until you restart the ATG platform	Changes to the value take place immediately, but are not retained if you stop the component

Note: If you are configuring a live component and change properties that are referred to by another component, the references are not updated until you restart the application; they are not updated when you stop or restart the component. For example, Component A has a status property, the value of which is linked to the status property of Component B, changes to the value of the Component B status property are not reflected in Component A. Stopping or restarting a referenced component leaves the application in an unstable state, and is not recommended.

Editing options depend on the type of property:

- String values provide a text field for editing. You can type values directly into this field or click the **...** button to open a pop-up editing window.
- The `int`, `long`, `float`, and `double` values provide a number field for editing.
- Boolean values provide a pull-down list with true/false options.
- Enumerated values provide a pull-down list of options.
- Array, hash table, and component values have a **...** button that opens a corresponding pop-up editing window.
- All property types have a @ button that lets you set the property value by linking to another component or component property.

In the case of our example, the port number for the RMI server is set by the `rmiPort` property (of type `int`). To change the port number, click in the value cell and type the new port number.

After you make changes, choose **File** > **Save** in the Component Editor window. If the component is live, a dialog box appears, asking if you want to copy your configuration changes to the live state. If you copy the changes, restart the ATG application to ensure that the changes take effect.

Changing Component Properties Manually

As an alternative to using the ACC or Configuration Manager, you can always edit properties files manually. A few configuration properties can only be configured manually, and are not accessible through the ACC or Configuration Manager.

Note, however, that when configuring properties manually, no errors are generated if you specify a property name incorrectly. The component may generate an error if it cannot find the value; in this case, check your properties file for typos.

To manually edit a properties file, do the following:

1. Create a new properties file in `<ATG10dir>/home/localconfig` with the same name and path structure as the original file. For example, the `defaultFrom` property in the `/atg/dynamo/service/SMTPEmail` component specifies the e-mail address from which messages will be sent via SMTP. To modify `defaultFrom`, create a new file called `SMTPEmail.properties` in the path `<ATG10dir>/home/localconfig/atg/dynamo/service`.

   Note: Step 1 is not necessary for the `Configuration.properties` file because a file of this name is created in the `<ATG10dir>/home/localconfig/atg/dynamo` directory during the installation process.

2. Add the desired property to the new file. For example, to change the setting for `defaultFrom`, such as to test@example.com, add the following line to the

`SMTPEmail.properties` file in
`<ATG10dir>/home/localconfig/atg/dynamo/service`:

```
defaultFrom=test@example.com
```

For example, to change the port number of ATG's RMI server to 8862 manually, open your `<ATG10dir>/home/localconfig/atg/dynamo/Configuration.properties` file and add (or modify) the following line:

```
rmiPort=8862
```

When specifying values for a property, you can add a manual line break using the backslash (\) line continuation character:

```
myList=valueOne,\
       valueTwo,\
       valueThree
```

This can help with readability when configuring lists of values.

Save the `Configuration.properties` file and restart the application. Because you made the change in the `localconfig` directory, the new port number will override the original value (still stored in the `config/atg/dynamo/Configuration.properties` file) and will be preserved when you install a new ATG platform distribution.

For additional information about defining and managing properties files, see the *Nucleus: Organizing JavaBean Components* chapter of the *ATG Programming Guide*.

Using Forward Slashes (/) and Backslashes (\)

When specifying values for file properties, Nucleus translates the forward slash (/) to the file separator for your platform (for example, Windows uses a backslash (\) as a file separator).

The backslash (\) is the escape character for properties files, so if you edit a properties file by hand, you must use two consecutive backslashes (\\) to specify a value that contains a backslash. For example:

```
documentRoot=\\WebServer6.1\\docs
```

The ACC Component Editor handles the escape character automatically; if you change properties using the ACC, use single backslashes.

Modifying Lists of Values

When adding a list of values to a property in a properties file, use the **+=** appending operator. This operator is commonly used in `localconfig/atg/dynamo/Initial.properties` to specify the components to create at startup time. For example:

```
initialServices+=/StartComps/services/comp1
```

The += operator specifies that you want to append /StartComps/services/comp1 to the value of initial services set elsewhere in the configuration path, rather than replace the value.

Similarly, you can use the -= operator to remove an item from a value list. This allows you to avoid redeclaring a list when you only want to remove one member. Note that in order for values to be removed, they must match exactly; if you specify 2.0 for removal, 2.00 is not removed. If the item to be removed is not found, no errors are generated.

Specifying Directory Paths

When you specify a directory path as a value in a component, you can do so either relative to the <ATG10dir>/home directory, or relative to your ATG server's directory.

Adding Comments to Properties Files

To add comments to a properties file that you've edited manually, you must add the comment in the $description field. If you preface the comment with a pound sign (#), the comment will be deleted if you subsequently modify the properties file using the ACC.

Using the Dynamo Component Browser

The Dynamo Component Browser, an element of the Dynamo Administration UI, is a window into ATG's Nucleus framework. From the Component Browser, you can view and modify components in a running Nucleus-based application.

To open the Component Browser, connect to the Administration UI using this URL:

```
http://hostname:port/dyn/admin
```

Enter your username and password; the defaults are admin and admin. When the Administration UI opens, click the **Component Browser** link.

The following topics are covered in this section:

- Component Browser Structure
- Changing the Running Configuration
- Starting Nucleus Components
- Customizing the Interface

Component Browser Structure

The Dynamo Component Browser is set up so that you can view and edit component properties. The Component Browser main page shows a list of components (called services in the Admin UI) currently running in Nucleus, such as Initial. When you click **Initial**, you see a page that shows the hierarchical location and class reference of that service:

```
Service /Initial/

Class atg.nucleus.InitialService
```

The forward slash character (/) separates the elements of the name into a hierarchy you can click through.

Note: Clicking the first forward slash (/) character brings you back to the main Nucleus service page.

Below the beginning information, you see tables with Properties, Event Sets, and Methods. The current service's property names and values are listed. Continuing on the `Initial` service page, if you click `loggingDebug` in the Properties table, you see a page that shows the properties of `loggingDebug`; you can edit these properties on this page. For example, to enable debugging errors to be logged, go to New Value and select **true**. Then click the **Change Value** button. To see the changes listed back on the `Initial` service page, click your browser's **Reload** button to refresh the view of the Properties table.

Note: Avoid changing system property values unless you know what they do. Changes set here will remain in effect while this ATG instance is running.

Changing the Running Configuration

You can change the configuration of a running Nucleus-based application from the Dynamo Component Browser. For example, on the `Initial` service page, click `loggingDebug` in the Properties table to see the properties of `loggingDebug`. To enable logging for debugging errors, go to New Value and select **true**, then click the **Change Value** button. To see the changes listed back on the `Initial` service page, click your browser's **Reload** button to refresh the view of the Properties table.

Note: Avoid changing system property values unless you know what they do. Values changed in the Dynamo Component Browser are not written to the properties files; when you stop and restart the application, configuration properties revert to those in the configuration properties file. To make permanent changes to configuration, make the change in development mode using the ACC, then redeploy the application.

Starting Nucleus Components

In addition to browsing for running components to change their configuration, you can use the Component Browser to start a Nucleus component that is not currently running. To start a stopped component, enter the full Nucleus path of the component in your browser. For example, you can start the `OrderRepositoryPipelineDriver` by going to this URL:

```
http://hostname:portnumber/dyn/admin/nucleus/atg/reporting/datawarehouse/
loaders/OrderRepositoryPipelineDriver
```

Customizing the Interface

By default, the Dynamo Component Browser displays a component by listing its contained children and the values of the component's properties. You might want to customize a component's administrative interface, for example to show more information about a service. To do this, override the methods in the default administrative servlet, `atg.nucleus.ServiceAdminServlet`. The Scheduler service, for example, extends the standard administration servlet to show information about all the tasks the scheduler is running. To see a list of these tasks, go to the following URL:

```
http://hostname:port/dyn/admin/nucleus/atg/dynamo/service/Scheduler
```

To customize an administrative interface, create a subclass of `atg.nucleus.ServiceAdminServlet`. For more information, see the *Nucleus: Organizing JavaBean Components* chapter of the *ATG Programming Guide*.

Common Configuration Changes

This section outlines several common configuration changes.

Modifying Environment Settings

ATG's startup behavior is affected by its CLASSPATH, the Java arguments passed to the Java Virtual Machine, and any custom environment variables you define. You can modify the startup behavior of these parameters as follows:

- CLASSPATH: ATG's `CLASSPATH` includes a `<ATG10dir>/home/locallib` directory, which you can use for any Java class files you create (classes should be in exploded form). Any classes stored in this directory are picked up by ATG automatically. For more information, see the *Nucleus: Organizing JavaBean Components* chapter of the *ATG Programming Guide*.
- Java Arguments: You can set or add to the arguments passed to the Java Virtual Machine by setting the environment variable `JAVA_ARGS`.

To customize the `CLASSPATH` and `JAVA_ARGS` settings, as well as define custom environment variables, see your application server documentation.

The following table lists some common values for `JAVA_ARGS`:

Java Argument	Description
`-Djava.rmi.server.hostname=`*`IP_Address`*	Configures Java Remote Method Invocation (RMI) to export RMI objects on a particular IP address; for more information, see Starting the ATG Control Center in the *Running Nucleus-Based Applications* chapter
`-Djava.compiler=NONE`	Turns off the just-in-time compiler so that stack traces include full line number information
`-Xms`*`size`*	Minimum size of memory heap for Java Virtual Machine on startup
`-Xmx`*`size`*	Maximum size of memory heap for Java Virtual Machine

`-Xnoclassgc`	Prevents garbage collection of classes
`-verbose[:class|gc|jni]`	Enables verbose output about each class loaded, garbage collection, or Java Native Interface (JNI) messages

For more information about arguments you can use with the `java` command, enter the command `java -help`.

Note: When setting `CLASSPATH` be careful to append or prepend your values onto the original value of the environment variable rather than replace it, or you will omit directories that ATG needs to start properly.

Modifying Custom Module Resource Settings

If you create a custom module (see the *ATG Programming Guide*), you can use the module's `MANIFEST.MF` file to specify paths to the module's resources, as follows:

- `ATG-Class-Path`: Specify a space-delimited set of paths to module resources that contain classes required by the module. For example:

  ```
  ATG-Class-Path: lib/resources lib/classes.jar
  ```

  ATG adds the `ATG-Class-Path` value to the `CLASSPATH` as each module is processed.

- `ATG-Config-Path`: Specify a space-delimited set of paths to module resources that provide Nucleus configuration files needed by the module's server application components. For example:

  ```
  ATG-Config-Path: config/config.jar config/oca-ldap.jar
  ```

  ATG adds the `ATG-Config-Path` value to the configuration path.

  Note: The path names in a module's `ATG-Class-Path` and `ATG-Config-Path` settings are relative to the module's root, not to the `<ATG10dir>` install directory.

In the `MANIFEST.MF` file, the `ATG-Required` attribute specifies which modules the custom module requires to start up. `ATG-Required` ensures that a given module's manifest is processed *after* it processes all the modules that the module depends on. For example, if you want to place the `config` directory for your custom module after the DPS `config` directories in the configuration path, configure the attributes as follows:

```
ATG-Config-Path: config/
ATG-Required: DPS
```

Enabling checkFileNameCase on Windows

In order to prevent Nucleus from creating new components unnecessarily during development, you can configure ATG to check the case of file names by setting the `checkFileNameCase` property of the `Nucleus` component to `true`. This prevents Nucleus from creating new components if, for example, you create a component named `Person` and then mistakenly refer to it as `person`.

The checkFileNameCase property has no effect on UNIX platforms. It imposes a small performance cost on Windows. Therefore, once your application is no longer in active development and you are not creating new components often, you should set the checkFileNameCase property back to false (the default).

The recommended deployment configuration (false) is set in the liveconfig configuration layer. To learn more about liveconfig settings, see Enabling liveconfig Settings in the *Configuring for Production* chapter.

LogListeners

ATG's global configuration settings are configured in the GLOBAL.properties file (located in config/config.jar). The settings in this file control logging and log listeners and apply to all ATG components in the config tree except those that set these properties explicitly themselves. If you want to edit this file, you must edit it manually.

The components listed in the logListeners property receive messages from components that send log events. By default, two log listeners are set: ScreenLog and LogQueue. ScreenLog writes messages to the console, while LogQueue puts messages into the log files.

```
logListeners=\
     atg/dynamo/service/logging/LogQueue,\
     atg/dynamo/service/logging/ScreenLog
```

On JBoss, the ScreenLog component is an instance of the CommonsLoggingLogListener class, which logs via the Apache commons logging APIs.

Normally, commons logging uses the class name of the class doing the logging. ATG has changed this slightly to provide the component's Nucleus path, prefixed with nucleusNamespace and separated by periods. The prefix prevents collisions with actual class names, and makes it clear that the logging component is a Nucleus component.

For example, the /atg/dynamo/Configuration component would have a commons logging classname of nucleusNamespace.atg.dynamo.Configuration. By default, you will see the short name of the component in the JBoss log (Configuration, in this example). To see the entire Nucleus path, set the useFullPaths property to true. The logging system will then print out atg/dynamo/Configuration as the short class name.

To disable global logging to the console, set the loggingEnabled property of the ScreenLog component to false.

See the *Logging and Data Collection* chapter of the *ATG Programming Guide* for more information.

Creating Additional ATG Server Instances

ATG server is the term for a specific collection of configuration information, which can then be included with your Nucleus-based application when you assemble the EAR file. It can include information such as machine names and ports, system paths, and connection pools.

The ATG platform installation comes configured with a default server instance in the `<ATG10dir>/home/servers/original` directory. You can create additional, individually configurable ATG servers by running the `<ATG10dir>/home/bin/makeDynamoServer` script, or through the Configuration Manager in the Dynamo Administration UI when the default server is running. If you are using CIM to configure your installation, CIM creates ATG servers for you (see Using the Configuration and Installation Manager (CIM) in this guide).

For information about assembling an EAR file that uses a non-default server, see *Using a Non-Default ATG Server* in the *Developing and Assembling Nucleus-Based Applications* chapter of the *ATG Programming Guide*.

Using the MakeDynamoServer Script

Run the `makeDynamoServer` script with the following syntax:

```
makeDynamoServer.bat new_server_name rmi_port_number drp_port_number
```

This script creates a new `<ATG10dir>/home/servers/new_server_name` directory with the following subdirectories and properties files:

```
|--- data
|--- j2ee
       |--- runtime
|--- localconfig  (includes CONFIG.properties)
       |--- atg
             |--- dynamo  (includes Configuration.properties)
|--- logs
       |--- archives
|--- pagebuild
|--- sessionswap
```

It sets the name property in the `localconfig/CONFIG.properties` file. For example:

```
name=Server myServer
```

It also sets the `rmiPort`, `rmiEnabled`, and `drpPort` properties in the `localconfig/atg/dynamo/Configuration.properties` file. For example:

```
rmiEnabled=true
rmiPort=9001
drpPort=9002
```

The DRP port value uniquely identifies the instance; the port itself is not used for communication.

Using the Configuration Manager

To open the Configuration Manager, connect to the Dynamo Administration UI using this URL:

```
http://hostname:port/dyn/admin
```

Enter your username and password; the defaults are `admin` and `admin`. When the Dynamo Administration UI opens, click the **Configuration Manager** link to see your configuration options.

To add a new server, click **Add, Delete, or Reset Servers**. Unless you explicitly set its properties, the new server inherits the properties of the `original` default server.

Configuring a New Server Instance

The Configuration Manager's server list shows the ATG servers registered with the Configuration Manager. Any changes you make to the *default configuration* affect all ATG servers that are using the default configuration for that setting.

To configure an individual server, click the server's name in the list. To configure a cluster, see the Configuring for Production chapter.

The Changing Component Properties with the ACC section includes an example of how to change the port number of ATG's internal RMI server. To make that same change using the Configuration Manager, do the following:

1. Click the name of the server you want to configure (for example, **Default Configuration**).

   The Server page opens, listing the configuration properties that you can modify in various categories.

2. In the Configure Internal Servers section, click the **RMI Server** link.
3. When the Configure RMI Service page opens, type the new port number in the RMI service port field.
4. Click **Apply Changes**.

The change is written to a properties file in your ATG installation, but does not affect the currently running Nucleus-based application. For a development-mode application, restart the application for the change to take effect. For a standalone application, reassemble and redeploy the EAR.

Setting Up a Configuration Group

A configuration group provides a mechanism for ensuring consistent configuration among ATG server instances. At startup, instances that are members of a configuration group download group configuration properties from the group's master server. At runtime, group members can periodically download updates that pertain to their group.

Note: Like other configuration changes, group configuration changes generally take effect only on instance startup; they have no effect on a running Nucleus component.

In order to join a group, an ATG instance must define itself as a group client or server by setting a `ConfigurationClient` or a `ConfigurationServer` component:

- A `ConfigurationClient` component obtains its group configuration settings from an ATG server instance that is designated as the group master. Each `/atg/dynamo/service/groupconfig/ConfigurationClient` component is an instance of this class:

  ```
  atg.service.configuration.group.ConfigurationClient
  ```

- A `ConfigurationServer` component maintains group configuration settings and ensures that those settings are uniform among all group members. Each `/atg/dynamo/service/groupconfig/ConfigurationServer` component is an instance of this class:

  ```
  atg.service.configuration.group.ConfigurationServer
  ```

One `ConfigurationServer` is designated as the default group master. Changes to group settings must be set on the master `ConfigurationServer`; it then distributes those changes to other `ConfigurationServers` and `ConfigurationClients` in the group.

A group can have one or more `ConfigurationServers`. If the primary master fails, another `ConfigurationServer` assumes the role of group master until the primary master resumes operation. The order of succession is established by the primary master and distributed to other `ConfigurationServers`.

Note: An ATG instance that serves as a configuration server can also act as a configuration client, and typically does so.

Requirements

To use group configuration, the following requirements apply to each ATG server instance in the group:

- The instance must be assembled with the `DafEar.Admin` module. Its`/atg/dynamo/Configuration.adminPort` property must be set to the port where the HTTP server is listening and can service the Dynamo Administration UI (`http://`*`host`*`:`*`port`*`/dyn/admin/`).
- For each ATG server instance, set its configuration group properties in the `Configuration.properties` file, as described later in this section.

Group Identifiers and Node Types

A configuration group is identified by its group name, where each `ConfigurationClient` and `ConfigurationServer` in the group is configured with the same `Configuration.groupName` property. Settings that are specific to a group are known only to the member ATG instances. An ATG instance can belong to only one group at a time.

Note: A configuration group can overlap multiple ATG server clusters — for example, publishing and production clusters.

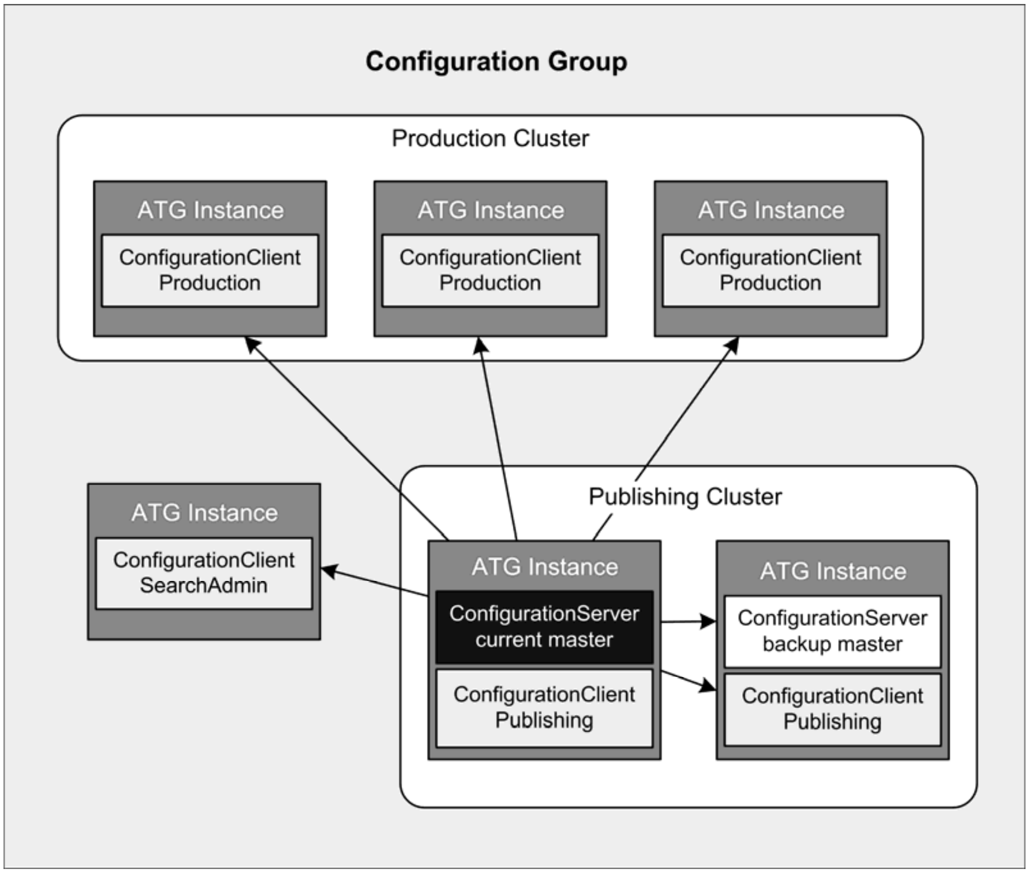
Within a group, ATG server types are differentiated through their node types. For example, a configuration group might contain these servers:

- Commerce production
- Commerce staging
- Search Merchandising
- Search servers
- Asset management

In order to differentiate settings among server types, each server's configuration client sets its `Configuration.nodeType` property to a value that corresponds to its server type. Given the previous server types, you might set their respective `Configuration.nodeType` properties as follows:

- `commerce-production`
- `commerce-staging`
- `search-merch`
- `search`
- `publishing`

The following diagram shows how a configuration group might be composed:


Configuration Group
Production Cluster
ATG Instance
ConfigurationClient Production
ATG Instance
ConfigurationClient Production
ATG Instance
ConfigurationClient Production
ATG Instance
ConfigurationClient SearchAdmin
Publishing Cluster
ATG Instance
ConfigurationServer current master
ConfigurationClient Publishing
ATG Instance
ConfigurationServer backup master
ConfigurationClient Publishing

Configuration Group Properties

In order to configure an ATG server instance to participate in a configuration group, the following properties must be set before startup in `Configuration.properties`, in one of the following locations:

```
<ATG10dir>/home/localconfig/atg/dynamo/service/groupconfig/

<ATG10dir>/home/servers/serverName/localconfig/atg/dynamo/
service/groupconfig/
```

Property	Type	Description
`groupName`	String	A unique string that defines the group. All group members must set this property to the same value.

Property	Type	Description
`defaultMasterServer`	boolean	Set to `true` for one ATG instance in the group, designates this configuration server to serve as the primary master. If set to `true`, the property `serverEnabled` must also be set to `true`. All other configuration servers in the group should set this property to `false`.
`serverEnabled`	boolean	Set to `true` for all primary and backup configuration server instances in the group.
`clientEnabled`	boolean	Set to `true` on every configuration client, enables an ATG instance to participate in a configuration group.
`clientNodeType`	String	Required for all enabled configuration client instances, associates a configuration client with settings that are specific to that node type. All configuration clients of the same node type must set this property to the same value.

For example, you might configure a master configuration server as follows:

```
groupName=myUniqueGroupName
defaultMasterServer=true
serverEnabled=true
clientEnabled=true
clientNodeType=generic
```

Optional Properties

You can also set the following properties in `Configuration.properties`:

Property	Type	Description
`autoDiscoveryEnabled`	boolean	Specifies whether auto-discovery is enabled.
`httpPort`	int	The HTTP port for this ATG instance.
`httpsPort`	int	The HTTPS port for this ATG instance.
`startingServerUrls`	URL[]	The array of starting servers. This property is required if your configuration group spans network subnets, or auto-discovery is disabled.

Configuration Server and Configuration Client Properties

You can set all required group configuration properties in `Configuration.properties`, as described earlier. If desired, you can fine-tune the behavior of configuration servers and configuration clients by setting their properties directly. For example, after detecting the failure of the master configuration server, by default a backup configuration server immediately assumes the master role or looks for another backup configuration server to assume that role. If desired, you can specify a latency period for a given configuration server by setting its `WaitBeforeBecomingServerTimeout` property.

Viewing Group Properties

At runtime, you can use the Dynamo Administration Component Browser to view the properties of all configuration client and configuration server components, in this Nucleus directory:

```
/atg/dynamo/service/groupconfig/
```

After the master configuration server collects all configuration properties from configuration clients in its group, you can review configuration errors by pointing the Component Browser at the master configuration server component. You can also review the settings that are currently in effect for the group.

Storing Group Configuration Files

Configuration files for a configuration group are stored in one of these directories:

- `<ATG10dir>/home/groupconfig`
- `<ATG10dir>/home/server/`*`serverName`*`/groupconfig` (for named ATG servers)

The `groupconfig` directory contains `server` and `client` subdirectories, which are used by the local `ConfigurationServer` and `ConfigurationClient`, respectively. The `client` directory obtains its content from the master `ConfigurationServer` and should not be edited. You should only update the `server` directory content on the master `ConfigurationServer`.

Node-Type Configuration

Configuration for a given node type is stored in the subdirectory of the same name. For example, if a configuration group defines two node types, `production` and `staging`, the master `ConfigurationServer` stores settings for them in two subdirectories as follows:

```
../groupconfig/server/nodetype/production
../groupconfig/server/nodetype/staging
```

For example, a `production` setting for `/atg/dynamo/service/jdbc/FakeXADataSource` is stored on the master `ConfigurationServer` in this directory:

```
$ATG_HOME/groupconfig/server/nodetype/production/atg/dynamo/service/jdbc/
FakeXADataSource.properties
```

This file is propagated to production clients as follows:

```
$ATG_HOME/groupconfig/client/nodetype/production/atg/dynamo/service/jdbc/
FakeXADataSource.properties
```

At startup, a client's `nodetype` directory is added to its configuration path and is read before its `instance` and `localconfig` directories.

Instance Configuration

Configuration settings that are specific to an ATG server instance can be stored on this path:

```
../groupconfig/client/instance/host-name+sever-name
```

The master configuration server stores configuration settings for each ATG server instance in this subdirectory:

```
../groupconfig/server/instance/host-name+sever-name
```

For example, a master configuration server maintains instance settings for server `production1` on host `saturn` in this directory:

```
../groupconfig/server/instance/saturn+production1
```

At startup, a server's `instance` directory is added to its configuration path and is read immediately before any `localconfig` property settings.

Note: In order to avoid ambiguity among instances on a given host, each instance subdirectory has its own `instance.id.properties` file. The properties in that file uniquely identify the given instance, so the master configuration server can differentiate among multiple ATG instances on the same host, if necessary.

Downloading Group Configuration

At startup, each configuration client and backup configuration server downloads the full `groupconfig` directory structure from the master configuration server. Local replication of all subdirectories enables a configuration client to start up in the absence of a configuration server, and to start as any of the defined node types.

For example, the layout of a `groupconfig` directory might look like this:

```
groupconfig
   server
    instance
        saturn+production1
        saturn+lockmgr
        jupiter+publishing1
        jupiter+publishing2
          ...
    nodetype
       production
       publishing
         ...
   client
```

```
instance
    saturn+production1
    saturn+lockmgr
    jupiter+publishing1
    jupiter+publishing2
      ...
nodetype
   production
   publishing
     ...
```

The master configuration server can be configured to create a group configuration JAR file as needed after startup, which other group members can download. To do so, set the master's `autoCreateConfigJars` property to `true`. You can also create a JAR file on the configuration server manually, by invoking one of these methods from the Dynamo Component Browser:

- `createGroupConfigJar()`
- `createGroupConfigJarIfNeeded()`

Each configuration client periodically checks the master configuration server for updates to the group configuration, according to the value set on its `ConfigurationClient.schedule` property. by default, every 60 seconds. You can also manually download updates to a client at any time by invoking its method `downloadConfigUpdate()`.

Finding a Group Configuration

Each configuration client caches information about known configuration servers. When required, it checks for configuration updates as follows:

1. Reads through its cached list of known configuration servers, starting with the last-known master configuration server.
2. If no previously known configuration server can be found from the cached list, uses Zeroconf—via the component `/atg/dynamo/service/jmdns/ClusterBroadcaster` —to find a member of its configuration group. It uses that member's published information to find the current master configuration server and downloads its configuration.
3. If starting up and all attempts to auto-discover a configuration server fail, starts up with the previously downloaded group configuration.
4. If starting up for the first time and no previous group configuration is available, logs an error message and starts up without it.

Auto-Discovery

As installed, the group configuration system uses Zeroconf to advertise the existence of ATG server instances in the configuration group. Configuration clients and configuration servers notify Zeroconf of their existence, which also compiles and maintains a list of the group's configuration servers and the order of master succession. Zeroconf maintains the following information about each group member:

Published Data	Description
`hostName`	Host name, included in broadcast messages
`port`	HTTP admin port, included as the port in broadcast messages
`httpsPort`	HTTPS admin port, if any
`serverDirectory`	ATG server directory of this instance, truncated to 255 characters (as required by the DNS Service Discovery specification)
`atgVersion`	ATG version string of this instance. For example: “2007.2”.
`groupName`	Name of the configuration group
`clientNodeType`	Node type of this configuration client
`isGroupServer`	boolean, specifies whether this instance is a configuration server
`isGroupClient`	boolean, specifies whether this instance is a configuration client
`isMaster`	boolean, specifies whether this instance is defined as the master configuration server
`commandLineModules`	The list of modules specified on the command line, truncated to 255 characters (as required by the DNS Service Discovery specification)

Validating Group Configuration Properties

A configuration group can be used to validate Nucleus properties across various configuration clients, through one or more configuration validators that run on the master configuration server. The master configuration server collects live property values from running configuration clients for validation. Validation errors and warnings are written out via Nucleus logging; these are also accessible on the master configuration server via the Dynamo Component Browser.

Installed Validators

All validators are registered with the component `atg/dynamo/service/groupconfig/validation/ValidatorRegistry`. The ATG distribution provides three validator components, in this Nucleus directory:

```
/atg/dynamo/service/groupconfig/validation/
```

Validator	Description
`UniquePortNameValidator`	Verifies that a configured port name is unique to a given host machine. This validator is useful for checking settings such as the `drpPort`, and can be used by ATG services to compose a unique ID for an ATG instance.

Validator	Description
`LiveConfigValidator`	Verifies whether all configuration clients of a given node type have the same `liveconfig` setting.
`RepositoryValidator`	Checks all known repositories to determine whether the following settings are consistent: - Client lock manager settings for repositories that use locked or distributedHybrid caching - Subscriber repository and event sender settings for repositories that use distributed caching

All registered validators are scheduled to run according to the value set on the property `ConfigurationServer.schedule` — by default, every 30 seconds. You can also manually execute all registered validators by invoking the `runValidators()` method on the master configuration server. Errors are logged every five minutes on the master configuration server.

Session Management in ATG Applications

This section discusses topics relating to session management in ATG applications running on third-party application servers.

The J2EE specification defines that each web application has its own session object and any attributes added to the session are only accessible from within that web application. The application server is entirely responsible for managing session life cycles; it generates a unique session ID, creates the session, invalidates it, fails it over, etc. An "ATG session" refers to session-scoped components. See the *ATG Programming Guide* for information on Nucleus component scopes. Also, keep in mind that Nucleus components have a tree structure, and can include multiple scopes, with each scope being rooted at a particular component. The root for session-scoped components is `/atg/dynamo/servlet/sessiontracking/GenericSessionManager/`*`sessionid/`* where *`sessionid`* is generated by the application server.

Sharing Session Information Among ATG Applications

You can run multiple ATG applications in the form of WAR files within a single EAR. In this case, you should share session-scoped Nucleus components so that your application will always have access to the same instance of session scoped components. By default, J2EE servers hand out different session objects in each web application visited, even if all requests came from the same browser. Sharing sessions across ATG applications ensures that you can build a J2EE application consisting of multiple WAR files in a single EAR, and each WAR has access to the same session-scoped components. Note that you should never run more than a single ATG EAR per application server instance.

When multiple web applications exist in the ATG EAR file, one of them must be designated as the parent application. Being the parent means that that application's session ID is used as the basis for creating the ATG session scope root.

By default, ATG makes the <ATG10dir>\DafEar\base\j2ee-components\atg_bootstrap.war file the parent web application. The parent context path is /dyn. No additional configuration is required to use this, but your web applications should define the atg.session.parentContextName and atg.dafear.bootstrapContextName parameters in their web.xml to point to the parent web-application as shown:

```
<context-param>
   <param-name>atg.session.parentContextName</param-name>
   <param-value>/dyn</param-value>
</context-param>
<context-param>
   <param-name>atg.dafear.bootstrapContextName</param-name>
   <param-value>/dyn</param-value>
   <description>The name of the DAF bootstrap WAR context.</description>
</context-param>
```

The context path the context-param points to must be for a WAR file with the SessionNameContextServlet defined in its web.xml:

```
<servlet>
   <servlet-name>SessionNameContextServlet</servlet-name>
   <servlet-class>atg.nucleus.servlet.SessionNameContextServlet
   </servlet-class>
</servlet>
```

Note that there can be only one parent web application specified per EAR file. Therefore, if you change the parent application, be sure to set the context-param to the same values in all web.xml files within your EAR file:

```
<context-param>
   <param-name>atg.session.parentContextName</param-name>
   <param-value>/portal</param-value>
</context-param>
```

Note: This information applies only to session-scoped Nucleus components, and does not affect HTTP sessions obtained using atg.servlet.ServletUtil.getDynamoRequest(request).getSession(), which retain a context particular to the current web application.

Session Interaction Outline

This section describes the request process and how a Nucleus session name context is associated with that request.

1. When a request comes in without a session ID in the cookie or in the URL, the application server creates a new session for the requested web application.
2. The ATG PageFilter determines if the session has been failed over and needs to be restored, or is a new session.

3. If the request is for the parent web application, a session name context is created with the current session ID and added to the Nucleus component `/atg/dynamo/servlet/sessiontracking/GenericSessionManager`.

   View that component in the HTML Nucleus Component browser to see a list of current ATG session name contexts and the web applications that share those name contexts.

   If the request is for a child web application, the parent application's session ID is resolved in one of two ways, depending on the application server.

   Some application servers maintain a single session ID between web applications for the same client (browser), in which case the session name context ID is the current web application's session ID. This behavior is controlled by the `/atg/dynamo/servlet/sessiontracking/GenericSessionManager.singleSessionIdPerUser` property, which is set to one of the following default values in the `DafEar` submodule configuration layer:

   - WebLogic – `false`
   - JBoss – `true`
   - WebSphere - `true`

   Note: Do **not** change these values from their defaults.

   When the `singleSessionIdPerUser` value is `true`, the application server uses the same session ID for all web applications, so lookup is not required. Note that the application server hands out the same session id, but **not** the same HttpSession object.

   When `singleSessionIdPerUser` is `false`, a lookup determines the session name context ID. This is done by the `atg.nucleus.servlet.SessionNameContextServlet servlet` (included in `atg_bootstrap.war`), using a `RequestDispatcher.include()` call. The `SessionNameContextServlet` does two things:

   - Sets the parent session ID as a request attribute that can then be used by the child web application to bind to the correct session context.
   - For application servers that don't allow request attributes to be shared between web applications, it also sets a cookie named `ATG_SESSION_ID` with the session ID. This behavior is controlled by the `/atg/dynamo/servlet/sessiontracking/GenericSessionManager.useSessionTrackingCookie` property, which is pre-configured with the correct value for each application server.

4. The `atg.parent.session.id` session attribute is set to the parent session ID to avoid repeating the lookup.

5. A new session-scoped context of type `atg.servlet.SessionNameContext` now exists under the `GenericSessionManager`. Because the ATG Nucleus components live outside the application server's session, an `atg.servlet.SessionBindingReporter` object is added to each web application session as an attribute. According to the J2EE spec, this object must be notified by the application server when the session is started (its `valueBound` method invoked) or invalidated (its `valueUnbound` method invoked).

6. The SessionBindingReporter increments a counter in the SessionNameContext it belongs to. This counter keeps track of the number of child web application session references to the Nucleus session scope. As each child is requested during the course of the browser session, this number increases.

7. When the application server expires a session, either because of a user request (session.invalidate() invoked) or due to a session timeout, it unbinds all the session attributes and invokes the atg.servlet.SessionBindingReporter.valueUnbound() method.

8. The valueUnbound decrements the SessionNameContext counter.

9. When the counter reaches 0, all the child and parent web application sessions have been expired and it is safe for the ATG Nucleus session scope to be removed.

Note: Because the only link to the underlying session is through the SessionBindingReporter attribute, session management is a common cause for memory leaks. One such leak occurs on IBM WebSphere in a clustered environment, where the session invalidation can occur in a different JVM instance than where the session originated. See the Session Management in a WebSphere Cluster section.

Managing User Sessions

You can manage user sessions from the Dynamo Component Browser for debugging or administrative purposes. To access the Session Manager, click through the hierarchy:

```
/atg/dynamo/servlet/sessiontracking/
```

Click **GenericSessionManager** to view sessions. Choose the selection criteria, then click the **View** button. Click an individual session to see its properties.

4 Configuring Databases and Database Access

Your ATG platform installation includes a preconfigured SOLID SQL database that contains the data necessary for running the ATG demo applications. The SOLID database is intended for evaluation and development purposes only. Before deploying your application in a production environment, you should configure both your application server and ATG products to use a production-quality database management system such as Oracle or Microsoft SQL Server. Your applications can then access application server resources and ATG components simultaneously. For a list of supported databases, see the Supported Environments page (http://www.atg.com/en/products/requirements/).

The ATG platform includes a number of tools for creating a database, adding and removing tables, configuring data sources and connection pools, and importing and exporting data. This chapter covers the following topics:

Note: Changing ATG's out-of-the-box database schemas is not recommended, although you can extend the schemas as necessary. If you do make any schema changes, you must migrate the changes manually when you upgrade to a new version of ATG.

Note: JBoss comes with its own demo database, Hypersonic (note the datasource `hsqldb-ds.xml` in the `/deploy` directory). Some JBoss components require that database, so do not remove it unless you also plan to remove those components.

Creating Database Tables Using SQL Scripts

The following sections explain how to create database tables for the ATG Adaptive Scenario Engine and ATG Portal.

- Creating Database Tables for ATG Adaptive Scenario Engine
- Creating Database Tables for ATG Portal

See the installation documentation for your other ATG products for information on creating database tables required for those applications.

Note: If you are using a utf8 Oracle database, before creating any tables, in order to avoid errors you must set the system `nls_length_semantics` to `char`:

```
alter system set nls_length_semantics=char;
```

Creating Database Tables for ATG Adaptive Scenario Engine

To create the database tables for the ATG Adaptive Scenario Engine, run the SQL scripts provided for the DAS, DPS, and DSS modules, as described in the following sections.

- Creating the DAS Tables
- Creating the DPS Tables
- Creating the DSS Tables

ATG Portal Note: The table creation scripts for ATG Portal also create the tables for the ATG Adaptive Scenario Engine; you do not need to create the DAS, DPS, and DSS tables separately. See Creating Database Tables for ATG Portal for details.

Creating the DAS Tables

To create the database tables in the DAS module, run the `das_ddl.sql` script from the following directory:

```
<ATG10dir>/DAS/sql/install/database-vendor
```

The `das_ddl.sql` script is derived from the subscripts listed in the table below. If necessary, you can run these subscripts individually from the following directory:

```
<ATG10dir>/DAS/sql/db_components/database-vendor
```

Script name	Purpose
`create_gsa_subscribers_ddl.sql`	Creates tables for event-listener registrations for distributed caching mode in the GSA
`create_sds.sql`	Creates a table for the switching data source service

`create_sql_jms_ddl.sql`	Creates tables for the Dynamo Message System
`create_staff_ddl.sql`	Creates the Dynamo Staff Repository for the GSA
`dms_limbo_ddl.sql`	Creates tables to store delayed JMS messages
`id_generator.sql`	Creates a table for managing ID spaces
`integration_data_ddl.sql`	Creates a table for storing caching information from the integration repository
`nucleus_security_ddl.sql`	Creates tables for Nucleus security data

Creating the DPS Tables

To create the database tables for DPS, run the `dps_ddl.sql` script from the following directory:

```
<ATG10dir>/DPS/sql/install/database-vendor
```

The `dps_ddl.sql` script is derived from the subscripts listed in the table below. If necessary, you can run these subscripts individually from the following directory:

```
<ATG10dir>/DPS/sql/db_components/database-vendor
```

Script name	**Purpose**
`logging_ddl.sql`	Creates tables for the logging and reporting subsystem
`logging_init.sql`	Initializes the logging and reporting tables
`user_ddl.sql`	Creates tables for the DPS schema

Creating the DSS Tables

To create the database tables for DSS, run the `dss_ddl.sql` script from the following directory:

```
<ATG10dir>/DSS/sql/install/database-vendor
```

The `dss_ddl.sql` script is derived from the subscripts listed in the table below. If necessary, you can run these subscripts individually from the following directory:

```
<ATG10dir>/DSS/sql/db_components/database-vendor
```

Script name	**Purpose**
`das_mappers.sql`	Creates tables used by sample mappers to record ATG startup and shutdown events

`dps_mappers.sql`	Creates tables used by sample mappers to record DPS events
`dss_mappers.sql`	Creates tables used by sample mappers to record DSS audit trail events
`scenario_ddl.sql`	Creates tables for the DSS Scenario Engine

Creating Database Tables for ATG Portal

The install file in the `<ATG10dir>/Portal/install/`*`database-vendor`* directory runs a set of scripts that create the required tables for the Portal Application Framework (PAF) and baseline gears.

Note: These scripts also create the tables for the ATG Adaptive Scenario Engine; you do not need to run the DAS, DPS, and DSS scripts separately. Note also that the install file uses the `<ATG10dir>/Portal/install/minimal-data.xml` file to create the minimum set of data structures necessary to run ATG Portal.

Use the following syntax to run the install file appropriate for your DBMS:

- `install-db2` *`userid password database`*
- `install-mssql` *`userid password host database`*
- `install-oracle` *`userid password database`*

The table creation scripts for ATG Portal are located in the following directories:

```
<ATG10dir>/Portal/paf/sql/install/database-vendor
<ATG10dir>/Portal/gear_dir/sql/install/database-vendor
```

Note: These scripts use an ATG-specific `JTDatasource` and `TransactionManager`, and cannot be used with your application server's data source or transaction manager.

Script name	Purpose
`alert_ddl.sql`	Creates tables for the Alerts Gear
`bookmarks_ddl.sql`	Creates tables for the Bookmarks Gear
`calendar_ddl.sql`	Creates tables for the Calendar Gear
`communities_ddl.sql`	Creates tables for the Communities Gear
`discussion_ddl.sql`	Creates tables for the Discussion Gear
`docexch_ddl.sql`	Creates tables for the Document Exchange Gear
`membership_ddl.sql`	Creates tables for storing membership requests
`paf_mappers_ddl.sql`	Creates tables used by sample mappers to record portal events

`Portal_ddl.sql`	Creates tables for the Portal Application Framework
`poll_ddl.sql`	Creates tables for the Poll Gear
`profile_ddl.sql`	Creates tables for storing profile data for personalized communities and pages
`soapclient_ddl.sql`	Creates tables for the Web Services Client Gear

Destroying Database Tables

The ATG platform includes SQL drop scripts for destroying database tables. (If you are using ATG Content Administration, see the *ATG Content Administration Programming Guide* for information on destroying the database tables for your content administration server.) Run the drop scripts in the reverse of the order used for table creation.

This section covers the following topics:

- Destroying Database Tables for ATG Adaptive Scenario Engine
- Destroying Database Tables for ATG Portal

Destroying Database Tables for ATG Adaptive Scenario Engine

This section covers the following topics:

- Destroying the DAS Tables
- Destroying the DPS Tables
- Destroying the DSS Tables

Destroying the DAS Tables

To destroy all DAS tables, run the `drop_das_ddl.sql` script from the following directory:

```
<ATG10dir>/DAS/sql/install/database-vendor
```

The `drop_das_ddl.sql` script is derived from the subscripts listed in the table below. If necessary, you can run these subscripts individually from the following directory:

```
<ATG10dir>/DAS/sql/uninstall/database-vendor
```

Script name	**Purpose**
`drop_dms_limbo_ddl.sql`	Destroys the tables used to store delayed JMS messages

`drop_gsa_subscribers_ddl.sql`	Destroys the tables for event-listener registrations for distributed caching mode in the GSA
`drop_id_generator.sql`	Destroys the table for managing ID spaces
`drop_integration_data_ddl.sql`	Destroys the table that stores caching information from the integration repository
`drop_nucleus_security_ddl.sql`	Destroys the tables for Nucleus security data
`drop_sds.sql`	Destroys the table for the switching data source service
`drop_sql_jms_ddl.sql`	Destroys the tables for the Dynamo Message System
`drop_staff_ddl.sql`	Destroys the Dynamo Staff Repository for the GSA

Destroying the DPS Tables

To destroy all DPS tables, run the `drop_dps_ddl.sql` script from the following directory:

```
<ATG10dir>/DPS/sql/install/database-vendor
```

The `drop_dps_ddl.sql` script is derived from the subscripts listed in the table below. If necessary, you can run these subscripts individually from the following directory:

```
<ATG10dir>/DPS/sql/uninstall/database-vendor
```

Script name	Purpose
`drop_logging_ddl.sql`	Destroys the tables for the logging and reporting subsystem
`drop_user_ddl.sql`	Destroys the tables for the DPS schema

Destroying the DSS Tables

To destroy all DSS tables, run the `drop_dss_ddl.sql` script from the following directory:

```
<ATG10dir>/DSS/sql/install/database-vendor
```

The `drop_dss_ddl.sql` script is derived from the subscripts listed in the table below. If necessary, you can run these subscripts individually from the following directory:

```
<ATG10dir>/DSS/sql/uninstall/database-vendor
```

Script name	Purpose
`drop_das_mappers.sql`	Destroys the DAS sample mapper tables

`drop_dps_mappers.sql`	Destroys the DPS sample mapper tables
`drop_dss_mappers.sql`	Destroys the DSS sample mapper tables
`drop_scenario_ddl.sql`	Destroys the DSS Scenario Engine tables

Destroying Database Tables for ATG Portal

The reset file in the `<ATG10dir>/Portal/install/`*`database-vendor`* directory runs a set of scripts that drop the database tables for ATG Portal. Use the following syntax to run the reset file appropriate for your DBMS:

- `reset-db2` *`userid password database`*
- `reset-mssql` *`userid password host database`*
- `reset-oracle` *`userid password database`*

Note: Once you run the reset file, you must run the install file again to use your database with the Portal Application Framework. See Creating Database Tables for ATG Portal for details.

The drop scripts for ATG Portal are located in the following directories:

```
<ATG10dir>/Portal/paf/sql/uninstall/database-vendor
<ATG10dir>/Portal/gear_dir/sql/uninstall/database-vendor
```

Note that the lines in these files that drop the DSS, DPS, and DAS tables are commented out by default as a safety measure. To drop those tables, uncomment the lines before running the script.

Script name	**Purpose**
`drop_alert_ddl.sql`	Destroys tables for the Alerts Gear
`drop_bookmarks_ddl.sql`	Destroys tables for the Bookmarks Gear
`drop_calendar_ddl.sql`	Destroys tables for the Calendar Gear
`drop_communities_ddl.sql`	Destroys tables for the Communities Gear
`drop_discussion_ddl.sql`	Destroys tables for the Discussion Gear
`drop_docexch_ddl.sql`	Destroys tables for the Document Exchange Gear
`drop_membership_ddl.sql`	Destroys tables for storing membership requests
`drop_paf_mappers_ddl.sql`	Destroys tables used by sample mappers to record portal events
`drop_portal_ddl.sql`	Destroys tables for the Portal Application Framework

`drop_poll_ddl.sql`	Destroys tables for the Poll Gear
`drop_profile_ddl.sql`	Destroys tables for storing portal profile data
`drop_soapclient_ddl.sql`	Destroys the tables for the Web Services Client Gear

Adding a JDBC Driver

To configure the ATG platform to use the JDBC driver for your DBMS, first install the driver software on your system as instructed by the manufacturer. See your application server documentation for information on where the driver should be installed.

Oracle users: Use the Oracle JDBC driver version that matches your Oracle server version. See the Supported Environments page (http://www.atg.com/en/products/requirements/) for supported database versions.

If you are using an Oracle OCI client to connect your application server to the Oracle database, the bit version of Oracle OCI client must match the bit version of your JDK. For example if your JDK is 32-bit, your OCI client should be 32-bit, regardless of your operating system bit size.

Oracle WebLogic users: WebLogic ships with an `ojdbc14.jar` located at `<WLdir>/wlserver_10.0.1/server/lib/`. More recent Oracle drivers may be available, in which case you should make certain that your CLASSPATH refers to the latest version, not the shipped version. Conflicts between Oracle's JDBC driver and Oracle's OCI native libraries result in crashes in the Oracle OCI driver.

Removing the SOLID JDBC Driver from the CLASSPATH

To remove the SOLID JDBC driver from the CLASSPATH, remove `solid/SolidDriver2.1.jar` from the following line in the `<ATG10dir>/DAS/META-INF/MANIFEST.MF` file:

```
ATG-Class-Path: lib/resources.jar lib/classes.jar solid/SolidDriver2.1.jar
```

Configuring ATG Data Sources for Data Import

ATG uses its own data sources when running data import scripts. These scripts are used for initial application configuration. The data source is based on `/atg/dynamo/service/jdbc/JTDataSource`, a Nucleus service that creates new connections to a particular database.

Your running ATG application will use your application server's native data sources (see Configuring Data Sources and Transaction Management in this guide).

J2EE JDBC supports the Java Transaction API (JTA) via the `javax.sql.XADataSource` interface. JTA allows multiple resources from different providers to participate in the same transaction. Using two-phase commits, data integrity across different services is ensured. ATG supplies a `DataSource` that sits on top of an `XADataSource` and returns wrapped Connections that are registered appropriately with the associated Transaction Manager. ATG's `DataSource` must get all its Connections from an `XADataSource`. Only a true `XADataSource` produces connections that behave properly in a two-phase commit controlled by JTA. `XADataSources` should be included in JDBC 2.0 drivers for the various database vendors.

The default `DataSource` connection pool, `JTDataSource`, uses the `FakeXADataSource` component, which is configured by default for the SOLID database. If you want to use a database other than SOLID, you must configure the desired connection pool properties, but note that this datasource should be used only to run ATG data import scripts.

You can set up and configure a connection pool manually by creating two files in your `localconfig/atg/dynamo/service/jdbc/` directory:

- *`connectionPoolName`*`.properties`
- *`connectionPoolName`*`FakeXA.properties`

where *`connectionPoolName`* is the name of the connection pool you want to create.

The *`connectionPoolName`*`.properties` file contains properties and values similar to the following:

```
$class=atg.service.jdbc.MonitoredDataSource
min=10
max=10
blocking=true
maxFree=-1
loggingSQLWarning=false
loggingSQLDebug=false
loggingSQLInfo=false
dataSource=/atg/dynamo/service/jdbc/<connectionPoolName>FakeXA
loggingSQLError=false
```

The `min` property determines the number of connections that the pool starts out with. The `max` property determines how many connections are to be kept around in the pool. When the pool starts, it immediately creates the minimum number of connections. Whenever a service requires a connection, it takes one from the pool. If there are no connections free, then the connection pool creates a new connection, until the maximum is reached. Due to various initialization calls, ATG requires at least three JDBC connections on install or when started with a new database. Setting the JDBC connection pool's `max` property to anything less causes ATG to hang when starting up.

If the maximum has been reached and a service requires another connection, then the service blocks until some other service frees up a connection. If the `blocking` property is set to `false`, then instead of blocking, the connection pool fails and results in a SQL exception.

The *`connectionPoolName`*`FakeXA.properties` file contains properties and values similar to the following:

```
$class=atg.service.jdbc.FakeXADataSource
server=localhost:1313
user=admin
needsSeparateUserInfo=false
URL=jdbc:solid://localhost:1313
readOnly=false
password=admin
database=
driver=solid.jdbc.SolidDriver
```

These properties tell the connection pool how to make a connection. The `driver` parameter specifies the name of the driver that should be used. The `URL` property specifies the name of the database server machine, the port of the database server (optional), and the name of the database on the server (optional). The format of the URL looks like this:

```
jdbc:driver name[:additional server information]
```

By default, the connection pool's driver and URL are configured for the SOLID database, as follows:

```
driver=solid.jdbc.SolidDriver
URL=jdbc:solid://localhost:1313
```

The `user` and `password` properties provide the connection with login access to the database, and must be recognized by the target database.

The `readOnly` property determines whether the resulting connection will only be used to perform read-only operations. Some drivers may be able to improve performance if this is true. Most applications require read and write access, so this property is usually `false`.

ATG wraps the `Connection` object to separate out SQL warning and info messages. This lets you see the SQL statements generated by ATG. It also catches `SQLExceptions` that occur on the connection and causes the connection to be closed when it is checked by into the resource pool. In addition to the standard ApplicationLogging log levels (`loggingError`, `loggingWarning`, `loggingInfo` and `loggingDebug`), a monitored connection lets you split off the SQL log messages with these properties:

Property	**Description**
`loggingSQLError`	logs SQL exceptions as errors
`loggingSQLWarning`	logs SQL warnings received by the pool
`LoggingSQLInfo`	logs SQL statements sent by the pool
`LoggingSQLDebug`	logs JDBC method calls made by the pool

By default, ATG turns SQL warnings off since they tend to be informational messages, not warnings. If you want to log these messages, set `loggingSQLWarning` to true.

Configuring Data Sources and Transaction Management

When you first install the ATG platform, it is configured to use its own data sources (JDBC connection pools) and transaction manager. The data sources point to the SOLID demonstration database that is installed with ATG. This database contains the tables necessary for running ATG and the demo applications.

When you deploy your sites, you should reconfigure your installation to use the data sources and transaction manager that your application server uses. While the SOLID database is suitable for evaluation and development; it is not designed for a high-volume site, and most application servers do not have drivers to support it.

Note that data sources for all application servers should always use the `READ_COMMITTED` isolation level (on DB2, use the equivalent `CURSOR STABILITY`).

Configuring Data Sources for JBoss

ATG applications running on JBoss use a `JTDataSource` component, which should be configured to point to a JNDI reference to a `DataSource` component running in JBoss.

The ATG platform installation includes an XML file that contains the default configurations for all the data sources for each application, along with a JNDI name for each data source. The ATG installer copies this XML file into the JBoss `deploy` directory so that these data sources are started when JBoss starts.

Where to Configure JBoss Data Sources

You should configure your data source in the `localconfig`, `jbossconfig`, or equivalent named configuration layer. See “Managing Properties Files” in the *ATG Programming Guide* for information on application-server-specific and named configuration layers.

In order to use the `jbossconfig` directory:

- Modify the `MANIFEST.MF` file for the given ATG module to include the following property:

  ```
  ATG-JbossConfig-Path: jbossconfig
  ```

- Create a `jbossconfig` directory and put the properties files there.

Note: If JBoss configuration files are stored in the `ATG-3rdPartyConfig-Path` layer, you might see errors if you start up applications on other application servers, because the datasources are configured to point to JNDI names that are not set up on that application server. Datasource configuration files that are specific to JBoss should be in the `ATG-JBossConfig-Path` rather than the `ATG-3rdPartyConfig-Path` of those data source configurations.

Configuring New JBoss Datasources

To configure a new data source, go to the `<JBdir>\server\`*`server_name`*`\deploy\atg-solid-ds.xml` file. Rename the XML file to something appropriate (for example `atg-oracle-ds.xml`), bearing in mind that the filename must end in `-ds.xml`. Edit the following configuration settings:

```
JNDI name
URL
driver class
username
password
transaction isolation level
connection pool numbers
```

See your application server documentation for information on the available parameters. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <xa-datasource>
    <jndi-name>atgcore_ds</jndi-name>
    <track-connection-by-tx>false</track-connection-by-tx>
    <isSameRM-override-value>false</isSameRM-override-value>
    <min-pool-size>5</min-pool-size>
    <max-pool-size>100</max-pool-size>
    <blocking-timeout-millis>5000</blocking-timeout-millis>
    <idle-timeout-minutes>15</idle-timeout-minutes>
    <transaction-isolation>TRANSACTION_READ_COMMITTED</transaction-
isolation>
    <xa-datasource-class>oracle.jdbc.xa.client.OracleXADataSource</xa-
datasource-class>
    <xa-datasource-property
 name="URL">jdbc:oracle:thin:@otto.na.ad.atg.com:1521:ora10r2</xa-datasource-
property>
    <xa-datasource-property name="User">username</xa-datasource-property>
    <xa-datasource-property name="Password">password</xa-datasource-property>
    <!-- Uncomment the following if you are using Oracle 9i
    <xa-datasource-property name="oracle.jdbc.V8Compatible">true</xa-
datasource-property>
   -->
    <exception-sorter-class-name>
        org.jboss.resource.adapter.jdbc.vendor.OracleExceptionSorter
    </exception-sorter-class-name>
  </xa-datasource>
</datasources>
```

If you have changed the JNDI name, you must also change the name configured in the `<ATG10dir>/home/localconfig/atg/dynamo/service/jdbc/JTDataSource.properties` file:

```
$class=atg.nucleus.JNDIReference
JNDIName=JNDIDataSourceName
```

For example, `java:/ATGOracleDS`.

Note that if you are using a `WatcherDataSource`, this would be configured instead in a `DirectJTDataSource.properties` file.

Adding Database Class Files

If your database driver is located anywhere other than the server's lib directory (for example, `C:\jboss\jboss-eap-5.1\jboss-as\server\`*`atg_server`*`\lib`), you must edit `<JBdir>/bin/run.sh|bat` and add your database class files, such as Oracle's `ojdbc14.jar`, to the JBoss classpath. To do this, search for `$JBOSS_CLASSPATH` and just above it, create a line:

```
JBOSS_CLASSPATH=path_to_ojdbc14.jar
```

Rebuild and redeploy your EAR file.

Configuring Data Sources for WebLogic and WebSphere

To configure ATG to use data sources for WebSphere or WebLogic, override the default configuration of each ATG data source, replacing it with a pointer to a WebSphere or WebLogic data source.

For example, several ATG repositories use as their default data source the component `/atg/dynamo/service/jdbc/JTDataSource`, which is of class `atg.service.jdbc.MonitoredDataSource`. Rather than reconfiguring the repositories individually, replace the `JTDataSource` with a component of class `atg.nucleus.JNDIReference`, so that the "data source" that the repositories now point to is just a JNDI reference to a WebSphere or WebLogic data source. To do this, you create a `JTDataSource.properties` file that contains these lines:

```
$class=atg.nucleus.JNDIReference
JNDIName=java:comp/env/jdbc/ATGDatasource
```

where *`ATGDatasource`* is the JNDI name of the WebSphere or WebLogic data source. Put this file in `<ATG10dir>/home/localconfig/atg/dynamo/service/jdbc/`.

Configuring Data Sources for an Oracle RAC Cluster

If you use ATG Content Administration, you must configure data sources for the destination repositories that are used during deployment to staging and production servers. These data sources require special configuration if the following conditions are true:

- The target site database is set up as an Oracle RAC cluster with multiple nodes.
- The target site runs on WebLogic or WebSphere.

In this case, you must configure an Oracle RAC cluster so that all operations within a given transaction are directed to a single cluster instance:

1. Set up a database service that runs on a single instance in the production RAC cluster.
2. This RAC cluster instance and its database service must be referenced by the data sources of the destination repositories that Content Administration uses for deployment. To do this, configure the data sources so their JDBC URL is set as follows:

   ```
   jdbc:oracle:thin:@RAC-instance:port:dbservice
   ```

For detailed information about destination repositories and how they are used for deployment, see the *ATG Content Administration Programming Guide*.

Setting the Transaction Timeout on JBoss

The default JBoss transaction timeout is 300 seconds. This may be too short for your purposes, particularly if you have a large ATG Commerce catalog.

To increase the transaction timeout:

1. Go to the `<JBdir>/server/atg/conf/jboss-service.xml` file.
2. Change the `<attribute name="TransactionTimeout">300</attribute>` to a higher number.

Setting the Transaction Timeout on WebLogic

WebLogic will automatically roll back transactions that don't complete in a certain number of seconds. The default setting is 30 seconds, which may be too short for compiling certain complex pages, especially pages that embed many page fragments.

When you are developing an application, a page must be recompiled each time you change it. If your application includes complex pages (particularly if you are developing a portal with ATG Portal), you can avoid transaction timeouts by raising the timeout setting to 600 seconds. Before deploying the application on a production site, you should precompile all of the pages. You can then lower the timeout setting.

To change the setting, open the WebLogic Server Console, go to the JTA page for the domain ATG is installed in, and change the value in the Timeout Seconds field. ATG recommends setting the timeout to 120 seconds.

Setting the Transaction Timeout on WebSphere

WebSphere will automatically roll back transactions that don't complete in a certain number of seconds. The default setting is 120 seconds, which may be too short for compiling certain complex pages, especially pages that embed many page fragments.

When you are developing an application, a page must be recompiled each time you change it. If your application includes complex pages (particularly if you are developing a portal with ATG Portal), you can avoid transaction timeouts by raising the timeout setting to 600 seconds. Before deploying the application on a production site, you should precompile all of the pages. You can then lower the timeout setting.

To change the setting, go to Servers > Application Servers > *`server`* > Transaction Service in the console.

Setting the Isolation Level for Transactions in WebSphere

ATG applications require a `READ_COMMITTED` isolation level for transactions. The default isolation level in WebSphere using MS SQL and DB2 is `REPEATABLE_READ`. To prevent deadlocks, use the WebSphere Administration Console to set the isolation level in the `atg-bootstrap.war` of your Nucleus-enabled application to `READ_COMMITTED`. See your WebSphere documentation for instructions.

Datasource Debugging

This section describes the use of the `WatcherDataSource` class to debug data source problems. This feature is automatically available for all application servers.

Using Datasource Debugging

The default `JTDataSource` allows you to monitor and log data source information for debugging purposes. It does this using the `WatcherDataSource` class. A `WatcherDataSource` "wraps" another data source, allowing debugging of the wrapped data source. For example:

```
/atg/dynamo/service/jdbc/JTDataSource.properties
$class=atg.service.jdbc.WatcherDataSource
# The actual underlying DataSource.
dataSource=/atg/dynamo/service/jdbc/DirectJTDataSource
```

Note: Due to the potential performance impact, the features described here should be used only for debugging in a development environment. Do not use datasource logging in a production environment unless absolutely necessary.

To view all logged data from the `WatcherDataSource`, go to `/atg/dynamo/service/jdbc/JTDataSource` in the Dynamo Component Browser.

WatcherDataSource Configuration

The default `WatcherDataSource` configuration is:

```
showOpenConnectionsInAdmin=false
logDebugStacktrace=false
loggingDebug=false
monitored=false
loggingSQLError=true
loggingSQLWarning=false
loggingSQLInfo=false
loggingSQLDebug=false
```

This default configuration logs the following information:

- `currentNumConnectionsOpen`
- `maxConnectionsOpen`
- `numGetCalls`
- `averageGetTime`
- `maxGetTime`
- `numCloseCalls`
- `averageCloseTime`
- `maxCloseTime`
- `averageOpenTime`

- `maxOpenTime`

For additional debugging information, you can set the following properties to `true`:

- `showOpenConnectionsInAdmin`—Lists currently open connections, along with the amount of time they have been held open and the thread that is holding them open. This information is useful for identifying Connection leaks. If `logDebugStacktrace` is also true, then stack traces are displayed as well.

  Note: This momentarily prevents connections from being obtained or returned from the DataSource, and severely affects performance.

- `loggingDebug`—Logs debug messages on every `getConnection()` and `close()` call. These messages include interesting information such as sub-call time, number of open connections, and the calling thread. If `logDebugStacktrace` is also `true` then a stack trace is logged as well.

- `logDebugStacktrace`—Creates stack traces on each `getConnection()` call. This allows the calling code to be easily identified, which can be useful when trying to find Connection leaks, code that is holding Connections open for too long, or code that is grabbing too many Connections at a time.

  Note: This is done by generating an exception, which affects performance.

- `monitored`—Gathers additional connection statistics and SQL logging.

Using the JDBC Browser

The Dynamo Administration UI includes a JDBC Browser (`http://`*`hostname:port`*`/dyn/admin/atg/dynamo/admin/en/jdbcbrowser/`) that enables you to examine the metadata of a database, including a listing of the tables, columns, and supported data types. The JDBC Browser also allows you to create tables, drop tables, execute queries, and examine the results of those queries.

All these operations are performed on a generic JDBC driver connection, meaning that the JDBC Browser should work with all databases for which a JDBC driver exists.

Configuring the JDBC Browser

The JDBC Browser obtains its JDBC connections from a JDBC connection pool service. By default, the service is set to the standard connection pool at `/atg/dynamo/service/jdbc/JTDataSouce`. This connection pool determines which JDBC driver and database to use.

If you want the JDBC Browser to use a different connection pool, modify the `connectionPool` property of `/atg/dynamo/admin/jdbcbrowser/ConnectionPoolPointer` so that it points to the desired connection pool service, using the following form:

```
/atg/dynamo/service/jdbc/your-pool-name
```

Create Table Operation

The Create table page provides a simple way for you to define a table and create it in the database. You can fill in the names and types of up to 10 columns in the table (any columns you leave blank will not be put into the table). The column types are expressed in JDBC types, which may or may not correspond directly to your database's data types.

The `Nullable`, `Unique`, and `Primary Key` flags indicate properties of the column. You'll have to be careful to avoid illegal combinations; for example, most databases do not allow a primary key to be nullable.

The `Additional Constraints` are passed straight through to the `CREATE TABLE` statement. This allows you to enter additional constraints, such as foreign keys or indices.

Drop Table Operation

The Drop table page drops the table you name.

Execute Query Operation

The Execute query page allows you to enter an arbitrary SQL statement that is passed through the driver to the database. The results of the statement are displayed in response. If the statement generates multiple result sets and update counts, all of those result sets and update counts will be displayed.

The flag marked `Show resulting column headings in long form` indicates whether extra result set metadata should be shown with each column. This tends to be rather extensive and is probably not necessary for most operations.

When you submit the query, you can submit with a commit or submit with a rollback. These options are only meaningful if `autoCommit` is `false`. If `autoCommit` is `true`, then the query will always be followed by a commit. The `autoCommit` property is set in the connection pool service.

Metadata Operations

All JDBC drivers provide metadata about the database that can be accessed through the JDBC interface. This metadata includes runtime information such as a listing of the tables and columns in the database. The metadata can also include information about the specific dialect of SQL that is understood by the database.

The JDBC Browser allows you to access all the metadata provided by the JDBC driver. Each of the metadata operations will first ask you to provide parameters for the requested metadata. For example, the `List tables` operation will ask for a `Catalog Name`, `Schema Name`, and `Table Name`. You can leave these fields blank, in which case all the appropriate metadata will be returned.

Using ATG Products with an IBM DB2 Database

To use a DB2 database, you must set the parameterizedSelect and useSetBinaryStream properties of the /atg/dynamo/messaging/SqlJmsProvider component to false.

In order for some import scripts to work, you must also set the following in your <ATG10dir>/home/localconfig/GLOBAL.properties file:

```
handleRangesInMemory=true
localTransactionModeInitialization=false
```

Create at least three tablespaces and bufferpools: one tablespace/bufferpool with a page size of 4KB, one with a page size of 16KB, and one with a page size of 32KB. See your DB2 documentation for more information. ATG recommends that you create more than one tablespace in each size; the number will vary depending on your data.

The db2_jms_procedures_ddl.sql file contains procedures that set the msgPollBatchSize property of SqlJmsProvider. The dms_topic_flag and dms_queue_flag procedures set a fixed batch size of 5000 (unlike Oracle or MSSQL, DB2 does not compute the batch size, but uses a fixed number).

If you find that the 5000-item configuration is not effective, you can change the setting and recompile the procedures using the following statements:

```
db2 connect to db2_alias user schema_owner_name using password
db2 -td@ -v -ffilename
```

For example,

```
db2 -td@ -v -fdb2_jms_procedures_ddl.sql > db2_jms_procedures_ddl.log
```

For web-based applications such as ATG, the recommended isolation level READ_COMMITTED. On Oracle and MSSQL, non-modifying transactions are allowed to read data while another transaction commits. DB2's treatment of this isolation level is different in two ways: first, it calls the isolation level CURSOR STABILITY, and second, it locks exclusively on a table that is being modified, preventing other transactions from reading data from those tables.

To modify DB2 so that it behaves the same way with CURSOR STABILITY that Oracle and MSSQL behave with READ COMMITTED, create the following registry entries for your database:

DB2_EVALUNCOMMITTED
Do not wait for uncommitted updates.

DB2_SKIPINSERTED
Do not wait for uncommitted inserts.

DB2_SKIPDELETED
Do not wait for uncommitted deletes.

Using ATG Products with a Microsoft SQL Server Database

ATG products do not support Unicode for MS SQL Server databases. To use Microsoft SQL Server with ATG products, be sure the `useSetUnicodeStream` property of all SQL repository components is set to `false` (default). To ensure that no ATG components are configured to use `useSetUnicodeStream`, you can set this property in your `localconfig/GLOBAL.properties` file:

```
useSetUnicodeStream=false
```

If you are creating localized content, set the `useSetAsciiStream` property to `false` in your `localconfig/GLOBAL.properties` file:

```
useSetAsciiStream=false
```

If you are using the Microsoft SQL Server 2005 JDBC driver, you must set `sendStringParametersAsUnicode` to `false` in your URL connection string. For example:

```
URL=jdbc:sqlserver://<SERVER>:<PORT>;databaseName=<DATABASE>;
sendStringParametersAsUnicode=false
```

The `sendStringParametersAsUnicode=false` setting avoids Unicode character conversion and enables MS SQL Server to use indexes in queries.

In addition, to prevent deadlocks and timeout problems, you must turn on `READ_COMMITTED_SNAPSHOT`. For example:

```
ALTER DATABASE <database_name> SET READ_COMMITTED_SNAPSHOT ON;
```

Using iNet (Merlia) Drivers

If you are using iNet drivers on JBoss, bear in mind that this driver does not allow for passing information by URL; therefore, some additional information must be set in the property fields, as shown in this example:

```
<xa-datasource>
  <jndi-name>ATGProductionDS</jndi-name>
  <track-connection-by-tx/>
  <isSameRM-override-value>false</isSameRM-override-value>
  <min-pool-size>5</min-pool-size>
  <max-pool-size>100</max-pool-size>
  <blocking-timeout-millis>5000</blocking-timeout-millis>
  <idle-timeout-minutes>15</idle-timeout-minutes>
  <transaction-isolation>TRANSACTION_READ_COMMITTED</transaction-isolation>
  <xa-datasource-class>com.inet.tds.DTCDataSource</xa-datasource-class>
  <xa-datasource-property name="ServerName">server_name</xa-datasource-property>
  <xa-datasource-property name="DatabaseName">database_name</xa-datasource-
property>
  <xa-datasource-property name="User">database_username</xa-datasource-property>
  <xa-datasource-property name="Password">database_password</xa-datasource-
```

```
property>
 <xa-datasource-property name="Mode">71</xa-datasource-property>
  <!-- sql to call when connection is created -->
  <new-connection-sql>select 1</new-connection-sql>
  <!-- sql to call on an existing pooled connection when it is obtained from
pool -->
  <check-valid-connection-sql>select 1</check-valid-connection-sql>

  <!-- corresponding type-mapping in the standardjbosscmp-jdbc.xml -->
  <metadata>
    <type-mapping>MS SQLSERVER2000</type-mapping>
  </metadata>
  </xa-datasource>
```

If you are using iNet drivers with WebLogic, when you create your data source, use the following settings:

- The type should be DataDirect's MSSQL type 4 XA.
- Set the following properties:
  - url—The full connection string for your data source.
  - driver—The driver name is `com.inet.tds.DTCDataSource`.
  - user—User name for the database account.
  - port—Connection port used for the database.
  - mode—This should normally be set to 71, as Unicode is not supported for MS SQL.
  - serverName—The machine name of the database host.
  - secureLevel—Set this to 0 if you are not using SSL. If you are using SSL, see your database documentation for information.

Moving Data from SOLID to the Production Database

If you want to move data from your SOLID database to the database used by your application server, you can do this using the `startSQLRepository` script. This script is described in detail in the *ATG Repository Guide*. To use this script, follow these steps:

1. Set the `DYNAMO_HOME` environment variable to `<ATG10dir>/home`.
2. In the Dynamo Administration UI, create a new ATG server that uses data sources that point to the SOLID database. This is the default configuration for a new server.
3. Use the `startSQLRepository` script to export data from the SOLID database. Include in the `startSQLRepository` command the `–s` *`servername`* switch. For example, if the server you created in the previous step is called `server1`, you can export the data from all of the ATG repositories using this command:

   ```
   bin/startSQLRepository -s server1 -exportRepositories all all.xml
   ```

4. Use the `startSQLRepository` script to import data from the XML file (created in the previous step) into the database used by your application server. Use the `–s` switch to specify a ATG server that is configured to use a ATG data source that points to that database. For example:

   ```
   bin/startSQLRepository –s server_name -import all.xml
   ```

   Note that the ATG data source must use an ATG-supported database driver. See the http://www.atg.com/en/products/requirements/ for a list of supported database drivers.

Oracle users: Before importing the demo data, set the `useSetCharacterStream` property of all SQL repository components to `true` so that non-8859 characters are displayed correctly. You can set this property in your `localconfig/GLOBAL.properties` file:

```
useSetCharacterStream=true
```

Microsoft SQL users: In order to run the ATG demos with a Microsoft SQL database, you must configure the database to be case-sensitive. See your MSSQL documentation for instructions. Note that the Quincy Funds demo is not supported for MSSQL.

Transferring the Demo Data

Use the commands in the following tables to transfer the Quincy Funds data from SOLID to the your production database.

Note: The Quincy Funds demo is supported only on SOLID and Oracle.

Exporting the Demo Data from SOLID

Use the command below (on one line, with no line breaks) to export the demo data from the SOLID database to an XML file called `all.xml`.

Demo Application	Command
Quincy Funds	`bin/startSQLRepository –s server_name -m DSSJ2EEDemo -exportRepositories all all.xml`

Importing the Demo Data to the Production Database

Use the command below (on one line, with no line breaks) to import the data contained in `all.xml` to the database used by your application server.

Demo Application	Command
Quincy Funds	`bin/startSQLRepository –s server_name -m DSSJ2EEDemo -import all.xml –repository`

Copying and Switching Databases

In most situations, you'll want to make database changes on an offline copy of the database, rather than on the database that runs your live site. Making changes on the live site can cause errors or inconsistencies or might adversely affect the performance of your live site. The ATG platform includes copying and switching functionality that lets you copy databases, using the database vendor's native bulk copy tools, and switch your live site between two different databases.

This section includes the following topics:

- Database Copy Operations
- Creating a DBCopier Component
- Configuring the DBConnectionInfo
- Configuring the DBCopier
- Setting the Native SQL Environment
- Switching Databases
- Configuring a SwitchingDataSource
- Database Switching and Query Caching

For information about using the database Copy and Switch features for ATG Commerce, see the *ATG Commerce Programming Guide*.

Database Copy Operations

The procedure for copying a database includes three basic steps:

- exporting data out of the source database to an OS file
- deleting any data in the destination database
- importing data into the destination database from the OS file

The base class for the ATG database copying facility is `atg.adapter.gsa.DBCopier`. It is important to note that `DBCopiers` use vendor-specific bulk copy and SQL utilities for speed, rather than using JDBC. This is accomplished by executing these commands in separate processes.

If the native bulk copy program operates on one table at a time, the `DBCopier` imports table data in the order in which the tables are specified and deletes table data in the reverse order. Thus, if there are foreign key constraints among the tables, the copy operation can still work if the tables are specified in dependency order. The various subclasses of `DBCopier` implement copying for different database vendors, using different vendor tools.

To use a `DBCopier`, follow these steps:

1. Create a `DBCopier` component. See Creating a DBCopier Component.
2. Configure `DBConnectionInfo` components for your source and destination databases. See Configuring the DBConnectionInfo.
3. Configure the `DBCopier` component as described in Configuring the DBCopier.
4. Set the SQL environment variables as described in Setting the Native SQL Environment.
5. Run the `DBCopier` by invoking its `copy()` method. For an example, see the *ATG Commerce Programming Guide*.

Creating a DBCopier Component

The class from which you instantiate the `DBCopier` depends on the database you are using. The following are subclasses of `atg.adapter.gsa.DBCopier` and are in package `atg.adapter.gsa`:

DBCopier Subclass	Vendor	Vendor Program
`BcpDBCopier`	Microsoft	`Bcp`
`DB2DBCopier`	IBM	`export/import`
`OracleDBCopier`	Oracle	`exp/imp`
`SolidDBCopier`	Solid	`solexp/solload`

For more information about the `DBCopier` subclasses, see the *ATG API Reference*.

Configuring the DBConnectionInfo

The connection information about the source database (the database you are copying from) and the destination database (the database you are copying to) is maintained in a component of type `atg.adapter.gsa.DBConnectionInfo`. Create a `DBConnectionInfo` for each database and configure it with the following information:

Property	Description
`Server`	The name of the database server
`User`	A valid username to connect to the database
`Password`	A valid password for the username specified by the user property

Note: The `DBConnectionInfo` settings are not expressed in JDBC terms. The settings are the values of the connection parameters used by OS tools (such as `bcp`) when connecting to the specified database.

Configuring the DBCopier

Set the following properties of the DBCopier:

Property	Description
Source	The DBConnectionInfo that service holds connection information for the database to copy from.
Destination	The DBConnectionInfo that service holds connection information for the database to copy into.
Tables	A comma-separated list of the names of the tables in the source database to be copied. If the native bulk copy program operates on one table at a time, the DBCopier imports table data in the order in which the tables are specified and deletes table data in the reverse order.
Directory	The name of a scratch directory for SQL and data files used during the copy operation. This directory must exist before the copy is launched. It is strongly recommended that no other processes or facilities use this scratch directory, especially other DBCopier instances.
CleanupDirectory	Set this to true to delete the files in the scratch directory after the copy is performed. Defaults to false.

In addition to the above properties, which are common to all DBCopier classes, each of the DBCopier subclasses has the following properties you may need to configure.

BcpDBCopier

This DBCopier for MSSQL databases uses the bcp utility for copy data. Generally, you can use this copier with the default property settings, with one exception. You should set the BcpDBCopier's maxTextOrImageSize property to a value no smaller than the largest text or image column in the tables being copied. See your Microsoft documentation for details.

DB2DBCopier

This DBCopier for DB2 databases uses the DB2 export and import utilities. If you are running the DB2DBCopier on UNIX or any other operating system that uses "/" as a directory separator, set the useUnixStyleDir property of the DB2DBCopier component to true. If "\" is the directory separator, set the useUnixStyleDir to false. The DB2 export utility wants to store binary objects in their own files, so make sure that the directory property points to a location in which these files can be stored temporarily. See your DB2 documentation for details.

OracleDBCopier

This class is a DBCopier for Oracle databases. This copier uses the Oracle exp and imp utilities. You can configure OracleDBCopier to use direct path for exporting. To enable direct path for exporting, set the useDirectPathForExport property of the OracleDBCopier to true. This property is false by default.

See your Oracle documentation for more information on using direct path with the exp utility.

SolidDBCopier

This class is a `DBCopier` for SOLID databases. You should not need to change any configuration settings on the `SolidDBCopier`, other than the SQL environment, connection information, and the database and table names.

Setting the Native SQL Environment

`DBCopier` components use vendor-specific bulk copy and SQL utilities for speed, rather than using JDBC. Therefore, to use a `DBCopier`, the native SQL environment for the database in question must be set up **before** starting your ATG application. This is required by the vendor tools in the database software. To use a `DBCopier` component, you must set up the environment in which the JVM runs as specified in the database vendor documentation. You can add this environment information to your `<ATG10dir>/home/localconfig/environment.sh` or `environment.bat` file. For information about the settings for your database, see the documentation from your database vendor.

For example, for Oracle you should set your environment up to look something like this:

```
ORACLE_HOME=/oracle-directory
PATH=$PATH:$ORACLE_HOME/bin
ORACLE_SID=ora8
```

Switching Databases

In many database-dependent applications, you may want to make changes in an offline database and then switch over your live application so that the inactive database becomes the live database. ATG's switching facility is based on a class named `atg.service.jdbc.SwitchingDataSource`. You can use a `SwitchingDataSource` in place of a regular data source (such as `atg.service.jdbc.MonitoredDataSource`). The `SwitchingDataSource` can be switched between two or more underlying `DataSources`. All `DataSource` method calls are passed through to the `DataSource` specified by the `currentDataSource` property of the `SwitchingDataSource`. Note that each `DataSource` that the `SwitchingDataSource` points to must be of class `atg.nucleus.JNDIReference`, with a `JNDIName` property that points to an application server data source. See Configuring Data Sources and Transaction Management for more information.

The switching database is meant to complement the `DBCopier` components. For example, if you are using ATG Commerce, you would update an inactive database, switch your live site to that database, then copy the currently-active database to the inactive database using the database vendor's native bulk copy tools.

Note: Unlike `DBCopier`, ATG's switching facility is a JDBC mechanism.

To set up and use a database switching service:

1. Configure `DataSources` that connect to your live and inactive databases.
2. Configure a `SwitchingDataSource` component, as described in Configuring a SwitchingDataSource.

3. Configure the Repository components that use the `DataSources` to point to the `SwitchingDataSource`. Also set the Repository components' `selectiveCacheInvalication` property (see *Configure Selective Cache Invalidation* in the *ATG Content Administration Programming Guide*).

Important: If you have multiple independent ATG clusters that share a single `SDSRepository`, make sure each cluster uses a unique set of `SwitchingDataSource` names. Otherwise, the clusters will interfere with each other during the switching process.

Configuring a SwitchingDataSource

Set the following properties of the `SwitchingDataSource` component:

Name	Description
`initialDataSourceName`	The short name for the `DataSource` that should be used for the `currentDataSource` on the very first run. On subsequent runs, the initial `currentDataSource` is obtained from the state recorded in the `SDSRepository`.

`dataSources`	Set to a `ServiceMap` of `DataSources`. This property maps short names of `DataSources` to their Nucleus component path. The following example shows how you might set the `dataSources` property:
	`dataSources=FirstDataSource=\`
	`/atg/dynamo/service/jdbc/FirstDataSource,\`
	`  SecondDataSource=\`
	`/atg/dynamo/service/jdbc/SecondDataSource`

`repository`	Set with a reference to `/atg/dynamo/service/jdbc/SDSRepository`. This refers to the switching data source repository, which keeps track of which database the switching data source points to at any time.

This sample shows the default format of the switching datasource used by the product catalog in ATG Commerce:

```
$class=atg.service.jdbc.SwitchingDataSource
#
# A map from data source names to data sources
#
dataSources=\
     DataSourceA=/atg/commerce/jdbc/ProductCatalogDataSourceA,\
     DataSourceB=/atg/commerce/jdbc/ProductCatalogDataSourceB

#
```

```
# The name of the data source that should be used on startup
#
initialDataSourceName=DataSourceA

repository=/atg/dynamo/service/jdbc/SDSRepository
```

Database Switching and Query Caching

If you are using a GSA repository and set the `cacheSwitchLoadQueries` property of the `GSAItemDescriptor` to `true`, the query cache is loaded for a cache switch. If `false`, the query cache starts out empty after a cache switch.

5 Configuring for Production

The default configuration settings in your ATG installation are designed for evaluation and development. When your ATG application moves from development to live deployment, you should change some of these configuration settings as described in this chapter for better performance.

This chapter covers the following topics:

Enabling liveconfig Settings

The settings in the ATG base configuration layer are optimized for application development, but are not appropriate for a production environment. When you’re ready to deploy your Nucleus-based application in a production environment, enable the settings in the `liveconfig` configuration layer. This layer overrides many of the default configuration settings with values that are more appropriate for a deployed site. For example, the `liveconfig` configuration layer improves performance by reducing error checking and detection of modified properties files.

To enable liveconfig, you can use the -liveconfig argument for runAssembler (see the *Assembling Applications* section of the *ATG Programming Guide*), or add the following line to the WEB-INF/ATG-INF/dynamo.env file in the atg_bootstrap.war module of your EAR file:

```
atg.dynamo.liveconfig=on
```

JBoss Note: If you are using ATG Portals with JBoss, and you use the -liveconfig flag when you create your EAR file, you must also have a lock manager configured and running in order to create or edit communities. See the *Locked Caching* section of the *ATG Repository Guide* for information on lock management.

To disable liveconfig in an application in which it is currently enabled, either reassemble the application without the -liveconfig flag, or remove or set the liveconfig value to off in WEB-INF/ATG-INF/dynamo.env file in the atg_bootstrap.war module.

Customizing liveconfig Settings

You can add your own configuration files or directories to the liveconfig configuration layer in your ATG installation. It is best to put any such custom settings in a separate directory from the <ATG10dir>/*ATG_module*/liveconfig directories, in order to keep track of which liveconfig settings are ATG settings and which are your own custom settings. For instance, if you have a custom application module named MyModule, you could create a MyModule/liveconfig directory in your module, and include in that directory any configuration settings that you want to take effect when the liveconfig configuration layer is enabled.

To add an entry to the liveconfig configuration layer, include it in your module's manifest in an entry like this:

```
ATG-LiveConfig-Path: liveconfig
```

For more information, see the *Working with Application Modules* chapter of the *ATG Programming Guide*.

Disabling Checking for Changed Properties Files

Some disk access and memory allocation overhead can be eliminated by setting the configurationCheckMilliseconds property of the Nucleus component (with a Nucleus component path of /) to -1. This property controls whether or how often ATG rereads .properties files or .java files the next time that an instance of a given component is created (components with global scope are only created once per JVM, so this does not affect them; see the *ATG Programming Guide* for information on component scope). The default is 1000. This feature is useful during development, but we recommend disabling it once a site goes live for better performance. The value -1 disables the reloading of properties and .java files altogether.

Note: If you subsequently make changes to .properties files or .java files on your live site (which you generally should not do), you will need to restart your application server before changes are picked up. If you change property settings using the ACC, you may need to restart to fully register changes that may affect interdependent components.

The recommended configuration is enabled in the liveconfig configuration layer.

Disabling the Performance Monitor

The Performance Monitor (`/atg/dynamo/service/PerformanceMonitor`) can be used to gather statistics about the performance of specific operations in ATG components. However, this information gathering can itself have a negative effect on performance. Therefore, for deployment, disable the Performance Monitor by setting its `mode` property to 0:

```
mode=0
```

The Performance Monitor is disabled in the `liveconfig` configuration layer.

For more information about the Performance Monitor, see the *Monitoring Site Performance* chapter.

Adjusting the pageCheckSeconds Property

ATG's Page Processor compiles JHTML pages into `.java` files (JSP compilation is handled by your application server). The page processor, located at `/atg/dynamo/servlet/pagecompile/PageProcessor`, checks for new Java Server Pages that need to be compiled. You can improve performance by increasing the Page Processor's `pageCheckSeconds` property. The page compile servlet uses this property value to determine whether to check for new Java Server Pages that need to be recompiled. If a request occurs within this time interval (measured in seconds) for the same page, ATG will not check the date on the file. This improves performance in serving pages.

A value of 0 causes ATG to check for new pages on each request. The default value is 1. The `liveconfig` value is 60.

Changing the Default Cookie Hash Key

To make user cookies more secure and prevent users from using another user's profile by changing their cookie, the ATG platform includes a profile ID check cookie that it uses to validate the user's cookie. When you use secure profile cookies, ATG sends two cookies, named DYN_USER_ID and DYN_USER_CONFIRM. The DYN_USER_CONFIRM cookie is a hash of the user ID cookie. If the hashed DYN_USER_CONFIRM cookie does not match the user ID cookie, then the cookies are ignored and a new profile is used.

You may want to change the key that ATG uses to hash the cookie from the default value, so that your sites' cookies will be hashed with a different key from that used by other sites that run ATG. To change the secret key that ATG uses to hash the user ID cookie, edit the `cookieHashKey` property of `atg/userprofiling/CookieManager`.

Fine-Tuning JDK Performance with HotSpot

Oracle's Java HotSpot technology is available in a Client Virtual Machine (VM) and a Server VM. The default Client implementation can be considerably slower than the Server implementation, so ATG recommends using the Server JVM.

Configuring Repositories

On a production site, it is critical that your ATG repositories be configured properly to ensure that data storage and retrieval are performed accurately and efficiently. Poorly configured repositories can result in performance bottlenecks and errors. In particular, repository caches must be tuned properly to ensure that data is retrieved quickly and is up to date. This section discusses repository settings to configure on your sites.

Setting Cache Modes

The ATG SQL repository offers a choice of cache modes. When you have only a single ATG instance installed, you can use the simple cache mode with no problems, since there is no chance of two servers using inconsistent copies of a repository item due to caching. However, when you deploy multiple ATG instances, you need to choose an appropriate cache mode for each item descriptor used by your application. See the *ATG Repository Guide* for more information.

In particular, if your sites are running more than one ATG server, it is highly recommended that you use `locked` mode caching for the `individualScenario` item descriptor. See the *ATG Personalization Programming Guide* for more information.

Remember that if you use `locked` mode caching, you must also enable lock manager components. See Enabling the Repository Cache Lock Managers in this chapter.

Prepopulating Caches on Startup

ATG performance typically improves after an application has been running a while, because more requests can be satisfied from caches. Under some circumstances, it may make sense to prepopulate your caches, so that you get the benefit of the caches immediately. Note, however, that this benefit may come at the cost of slower startup times.

You can prepopulate caches in a SQL Repository by using `<query-items>` tags in a repository definition file. For more information, see the *ATG Repository Guide*.

Enabling the Repository Cache Lock Managers

If you are using a SQL Repository with locked mode caching, you must enable the `ClientLockManager` (`/atg/dynamo/service/ClientLockManager`) on each ATG server and enable the `ServerLockManager` (`/atg/dynamo/service/ServerLockManager`) on one or more ATG servers. You may want to dedicate an ATG server only to lock management. Note that elements of the ATG Adaptive Scenario Engine are configured in the `liveconfig` layer to use locked mode caching by default.

Add the `ServerLockManager` component to the `initialServices` property of the `/atg/dynamo/Initial` component, and make sure that the `ClientLockManager` points to the correct host. The `ClientLockManager` should be configured like this:

`lockServerAddress`	The hostname of the machine running the `ServerLockManager`

lockServerPort	The port configured in the ServerLockManager component (9010 by default)
useLockServer	True

The ClientLockManager is enabled by the liveconfig configuration layer. For more information about cache lock managers, see the *ATG Repository Guide*.

Configuring Repository Database Verification for Quicker Restarts

By default, each SQL Repository component verifies each of the tables in its database on startup with a simple SQL query. These verification queries can slow the ATG startup routine. There are several approaches you can take to modify the SQL Repository startup procedures that can result in dramatically faster start times. In particular, you may wish to set the updateSchemaInfoCache property to true in your atg.adapter.gsa.GSARepository components, such as /atg/dynamo/service/jdbc/ProfileAdapterRepository. For details, see the *SQL Types and Repository Data Types* section in the *SQL Repository Item Properties* chapter of the *ATG Repository Guide*.

Configuring a Content Distributor System

ATG includes a content distributor system that allows you to cache content from repositories to an HTTP server. Using this system can significantly speed up request handling on a site. By default, only ATG Commerce uses this system, but it can be used by any ATG application.

The content distributor system is described in the *ATG Programming Guide*. If you are using an HTTP server such as Apache, no additional configuration of the content distributor system is required. If you are using your application server as your HTTP server, however, you need to configure the system to prepend the context path of the atg_bootstrap.war application (by default, /dyn) to the URL of any file it sends to the server.

The class atg.distributor.DistributorSender has a property named documentRootContextPath that you can set to specify the string to prepend. For example, for the distributor system used by ATG Commerce, set this property in the component /atg/commerce/catalog/ContentDistributor, either through the ACC or by adding the following line to the properties file of that component:

```
documentRootContextPath^=/atg/dynamo/Configuration.defaultDynamoPrefix
```

Configuring Targeted E-Mail

When running on your application server, ATG's targeted e-mail system makes loopback requests back to the server to render the e-mail template for each e-mail recipient. ATG makes one loopback request to create an HTTP session, and uses that session's ID when making subsequent loopback requests to render the template.

Nucleus Components

The components /atg/userprofiling/email/TemplateEmailSender and /atg/scenario/IndividualEmailSender (both of class atg.userprofiling.TemplateEmailSender) have several properties used for configuring loopback requests. The following table lists these properties and their defaults when running ATG on your application server:

Property and Default	Purpose
siteHttpServerName^=/atg/dynamo/Configuration.siteHttpServerName	Server name for loopback requests.
siteHttpServerPort^=/atg/dynamo/Configuration.siteHttpServerPort	Port number for loopback requests.
applicationPrefix^=/atg/dynamo/Configuration.dynamoEarContextRoot	The context path of the application.
initSessionURL=/init-session	The URL pattern used by InitSessionServlet (see below).
sessionManager=/atg/dynamo/servlet/sessiontracking/GenericSessionManager	Used to find the session from the session ID.
loopbackRequestsEnabled=true	Determines whether loopback requests are performed. Can be set to false if you are using this TemplateEmailSender only with DSP templates (see below).
contextPathPrefix^=/atg/dynamo/Configuration.defaultDynamoPrefix	String to prepend to template URLs. Default is /dyn/dyn/.

If you are using JHTML templates exclusively, you can disable loopback requests by setting the loopbackRequestsEnable property to false. In addition, you should set the contextPathPrefix property to null, and set the setupLoopbackTemplateEmailRequests property of the /atg/dynamo/servlet/pipeline/DynamoServlet component to false.

Configuring Web Applications

To enable targeted e-mail in an application, the application must run an instance of atg.nucleus.servlet.InitSessionServlet. For example, the web.xml file for the atg_bootstrap.war application includes the following lines:

```
<servlet>
  <servlet-name>InitSessionServlet</servlet-name>
  <servlet-class>atg.nucleus.servlet.InitSessionServlet</servlet-class>
</servlet>
<servlet-mapping>
```

```
      <servlet-name>InitSessionServlet</servlet-name>
      <url-pattern>/init-session</url-pattern>
   </servlet-mapping>
```

This servlet handles the requests to `/dyn/init-session`, and, as its name implies, initializes a session.

Setting Access Levels for Properties Files

ATG components are configured with plain text properties files. You should set access levels on your properties files so they can't be altered or viewed by unauthorized users. Only site administrators should have **read** and **write** permission. ATG must be invoked from an account with these permissions as well. The properties files that contain sensitive information typically reside in each server's `localconfig` directory. The most important properties files to protect include:

Component	Description
`/atg/dynamo/Configuration.properties`	Basic configuration for ATG
`/atg/dynamo/security/BasicSSLConfiguration.properties`	Default configuration for any service that uses SSL
`/atg/dynamo/service/jdbc/FakeXADataSource.properties`	Distributed transaction DataSource
`/atg/dynamo/service/jdbc/JTDataSource.properties` **Note:** Multiple versions of this component may exist in your installation; all of them may contain information that should be protected.	JTA participating and pooling DataSource
`/atg/dynamo/service/POP3Service.properties`	Checks the POP server for bounced e-mail

The most important ATG Commerce properties files to protect include:

Component	Description
`atg/commerce/jdbc/ProductCatalogFakeXADataSourceA.properties`	A distributed transaction `DataSource`
`atg/commerce/jdbc/ProductCatalogFakeXADataSourceB.properties`	A distributed transaction `DataSource`

These ATG Commerce properties files are located in a .jar file at <ATG10dir>/DCS/config/config.jar. For more information on ProductCatalogFakeXADataSourceA.properties and ProductCatalogFAkeXADataSourceB.properties, refer to the *ATG Commerce Programming Guide*.

Setting Logging Levels

By default, ATG sends all log events to two log listener components: /atg/dynamo/service/logging/LogQueue (which directs output to log files) and /atg/dynamo/service/logging/ScreenLog (which directs output to the console screen). Logging to the screen can cause performance problems on a production site. You can disable logging to the screen by setting the loggingEnabled property of the ScreenLog component to false.

If you want to disable logging entirely, or specify different logging levels, you can do that in the GLOBAL.properties file. For example:

```
loggingError=true
loggingWarning=true
loggingInfo=true
loggingDebug=false
```

The loggingDebug log generates large numbers of messages, which can impair performance, so loggingDebug should be set to false on a live site. You still have the option of overriding the global settings for a specific component. For example, if loggingDebug is set to false in the GLOBAL.properties file, you can still enable it for an individual component by setting that component's loggingDebug property to true.

See the *Logging and Data Collection* chapter of the *ATG Programming Guide* for more information.

Limiting Initial Services for Quicker Restarts

When you restart an ATG application, it starts up the services specified by the initialServices property of the /atg/dynamo/Initial component. You may add services to this list while you are developing your application. These services may in turn start up other components. Starting up a service at the same time as ATG ensures that the service is created and ready when it is first called upon. However, if too many services are configured to start up at the same time, then the ATG startup routine can become time-consuming and server restarts may be slow, which might make it more difficult to recover and restart if a server runs into problems. If server startups seem to be taking too long, consider whether some services can be started up on some other schedule than immediately on ATG startup. See *ATG Programming Guide* for more information about the Scheduler service.

If you set the loggingInfo property of the Nucleus component (with a Nucleus path of /) to true, and then start up ATG, the resulting info messages display an indented list of when each service starts up. From this list, you can determine which component causes which other components to be started.

Disabling Document and Component Indexing

The ACC creates and maintains indexes of documents and components. For sites with large numbers of documents or components, indexing can take time and CPU resources. Once your sites are deployed and relatively stable, you may want to limit or eliminate the indexing of documents or components.

The document and component indexes are maintained incrementally once built, and are rebuilt completely once a day at 1 a.m. by default. An index is rebuilt at startup only if it does not exist at all.

You can selectively exclude portions of the document tree from indexing by adding absolute pathname prefixes to the `excludeDirectories` property of the `/atg/devtools/DocumentIndex` component. The same is true for component indexing, but the component is `/atg/devtools/ComponentIndex` instead. To improve performance on a live site, you can turn off all document and component indexing by setting the `enabled` property of the `DocumentIndex` and `ComponentIndex` components to `false`.

Enabling the ProtocolChange Servlet Bean

ATG includes a servlet bean named `/atg/dynamo/droplet/ProtocolChange`. The `ProtocolChange` servlet bean lets pages switch between secure and nonsecure HTTP servers. The `ProtocolChange` servlet bean takes a URL as input and renders a URL that uses either the HTTP protocol or the HTTPS protocol, depending on the output parameter specified. The default configuration is:

```
secureHost^=/atg/dynamo/Configuration.siteHttpServerName
nonSecureHost^=/atg/dynamo/Configuration.siteHttpServerName
securePort=443
nonSecurePort^=/atg/dynamo/Configuration.siteHttpServerPort
secureProtocol=https
nonSecureProtocol=http
enable=false
```

When the enable property is `false`, the servlet bean renders the URL without changing the protocol. To enable this servlet bean to change the protocol, set the `enable` property to `true`. Also, ensure that the `secureHost` and `securePort` properties are set to values appropriate for your sites.

Setting up Clustering on JBoss

A cluster is a set of JBoss servers working together, serving pages at the same port. From the user’s point of view, all of the servers function as a single server; it doesn’t matter which server handles a given request. JBoss documentation refers to a cluster as a partition.

Virtually all production sites use clustering. Clustering provides much better performance and reliability than running on a single server. For example, if one server in a cluster goes down, the user will not be aware of it, because the other servers in the cluster can take over the sessions it was handling.

Setting up clustering of JBoss servers running ATG applications involves the steps described in the following sections.

Configuring the HttpPort Property

When running ATG server instances in a JBoss cluster, you must configure the `httpPort` property in the `/atg/dynamo/Configuration.properties` component to match the port set in the `siteHttpPort` property. If this is not done, the ATG email sender will fail. For example:

```
siteHttpPort=8080
httpPort=8080
```

Creating ATG Servers

The first step is to create your ATG servers, using the Configuration Manager or the `makeDynamoServer` script (see Creating Additional ATG Server Instances in the Configuring Nucleus Components chapter).

A typical production environment includes: a server lock manager, process editor server, workflow process manager, etc., for services that require a dedicated server, plus several servers that handle page requests. The servers you need to create depend on which ATG applications you are using, and on your unique site requirements.

Assembling for a JBoss Cluster

When you assemble your application, the application assembler includes all of the ATG servers you have configured (see "Assembling Applications" in the *ATG Programming Guide* for information on application assembly). This means that you can build your application once for each JBoss partition, deploy it on each partition, and enable the appropriate ATG server on each instance simply by changing the value of the `atg.dynamo.server.name` system property when you start up JBoss:

```
bin\run or bin/run.sh -c server_name -Datg.dynamo.server.name=
ATG_server
```

To assemble and configure your ATG application to run on a JBoss partition, when you invoke the application assembler, use the `-liveconfig`, `-standalone`, `-distributable`, and `-pack` flags in `runAssembler` as in the example:

```
bin/runAssembler –liveconfig –standalone –distributable –pack
output_file_name.ear –m module-list DafEar.Admin
```

The `–pack` flag is optional. The `–distributable` flag is required to enable JBoss session failover. Do not use the `–server` flag to specify an ATG server configuration. If you are using a named configuration layer, specify that as well (see "Managing Properties Files" in the *ATG Programming Guide* for information on named configuration layers).

Creating and Configuring JBoss Servers

Create and configure your JBoss servers; see the JBoss documentation for configuration information.

1. Use the `<JBdir>/server/all` server as a template for creating JBoss instances, since it is set up for clustering.
2. Make configuration changes, such as removing unneeded JBoss services (see JBoss Application Framework Trimming in this guide).
3. Copy the `/all` server for each corresponding ATG server.

Deploying Your Application

See the JBoss documentation for information about deploying to JBoss clusters.

Setting Up Clustering on WebLogic

A cluster is a set of WebLogic servers working together, serving pages at the same port. From the user's point of view, all of the servers function as a single server; it doesn't matter which server handles a given request.

Virtually all production sites use clustering. Clustering provides much better performance and reliability than running on a single server. For example, if one server in a cluster goes down, the user will not be aware of it, because the other servers in the cluster can take over the sessions it was handling.

Setting up clustering of WebLogic servers running ATG applications involves the following steps:

1. Create a group of WebLogic servers for serving pages, and assign them to a cluster.
2. Create additional WebLogic servers for the ATG lock manager, process editor server, workflow process manager and any other services that require a dedicated server. Assign these servers to a different cluster from the page servers.
3. For each WebLogic server, create a corresponding ATG server configuration.
4. Assemble your ATG application, and deploy it on each WebLogic server in both clusters. Configure the application on each WebLogic server to use the ATG server configuration that corresponds to that server.

See the Oracle WebLogic documentation for information about creating WebLogic servers and clusters. For information about creating ATG server configurations, see Creating Additional ATG Server Instances in the Configuring Nucleus Components chapter.

Assembling for a WebLogic Cluster

When you assemble your application, the application assembler includes all of the ATG servers you have configured. This means that you can build your application once, deploy it on each WebLogic server, and enable the appropriate ATG server on each instance simply by changing the value of the `atg.dynamo.server.name` system property when you start up WebLogic.

Follow these steps to assemble and configure your ATG application to run on a WebLogic cluster:

1. When you invoke the application assembler, use the -standalone flag to assemble the application in standalone mode, so it is not dependent on your ATG installation.

   Note: You cannot use -pack and -standalone in combination on WebLogic.

   In addition, use the -liveconfig flag to enable the liveconfig configuration layer. Do not use the -server flag to specify an ATG server configuration.

   If you are using a named configuration layer, specify that as well (see "Managing Properties Files" in the *ATG Programming Guide* for information on named configuration layers).

2. Deploy the application on each WebLogic server.

3. On each WebLogic server, enable the corresponding ATG server configuration by creating the atg.dynamo.server.name property for the JVM the server is running on and setting the property to the name of the ATG server. For example:

   ```
   startManagedWebLogic.bat myWebLogicServer -
   Datg.dynamo.server.name=myserver
   ```

Clustering Example

Suppose you want to set up a site consisting of an Administration Server, three servers that serve pages, one server that runs the ATG lock manager, and one server that runs the process editor server. Here's an example of how you might do this:

1. Start up WebLogic Server using the startWebLogic script. This starts up the WebLogic Administration Server (default name myserver, default port 7001).

2. In the WebLogic Console, create a server named pageServer. Assign it port number 7700. Assign an IP address used by no other server in the domain.

3. Create a cluster named pageCluster. Put pageServer1, pageServer2, and pageServer3 into this cluster.

4. Create servers named procedit and lockmgr. Assign each server the port number 7800. Assign each server a unique IP address.

5. Create a cluster named serviceCluster. Put procedit and lockmgr into this cluster.

6. Assign the two clusters different multicast addresses.

7. Using either the Dynamo Administration UI or the makeDynamoServer script, create ATG servers named pageServer1, pageServer2, pageServer3, procedit, and lockmgr. (You do not need to give the ATG servers the same names as the WebLogic servers, but it is a good idea do so.)

8. Configure the ATG lockmgr server to run the ATG ServerLockManager. (See Enabling the Repository Cache Lock Managers for more information.)

9. Configure the ATG Scenario Manager to run the process editor server on the ATG procedit server. (See the *ATG Personalization Programming Guide* for more information.)

10. Set up ATG session backup, as discussed in Enabling Component Backup.

11. Assemble your application, deploy it on each server in both clusters, and configure each instance to use the ATG server corresponding to the WebLogic server the instance is running on. (This process is discussed in Assembling for a WebLogic Cluster.)

12. Undeploy any applications that are deployed on the Administration Server.

13. Configure your HTTP server to serve pages from each server in `pageCluster` (but **not** any of the other servers).

14. Shut down the Administration Server and then restart it. This will ensure that all of the changes you made will take effect.

15. Start up the managed servers you created, using the `startManagedWebLogic` script. The syntax of this script is:

    ```
    startManagedWebLogic WebLogicServer adminURL-
    Datg.dynamo.server.name=myserver
    ```

    where *WebLogicServer* is the name of the WebLogic server, and *adminURL* is the URL of the WebLogic Administration Server. Let's assume that the hostname for the Administration Server is `myMachine`. To start up the WebLogic `pageServer1`, the command would be:

    ```
    startManagedWebLogic pageServer1 http://myMachine/7001
    ```

Setting up Clustering on WebSphere

A cluster is a set of WebSphere servers working together, serving pages at the same port. From the user's point of view, all of the servers function as a single server; it doesn't matter which server handles a given request.

Virtually all production sites use clustering. Clustering provides much better performance and reliability than running on a single server. For example, if one server in a cluster goes down, the user will not be aware of it, because the other servers in the cluster can take over the sessions it was handling.

Installing and Configuring WebSphere

The first step in setting up a clustered deployment is to install and configure the WebSphere cluster. See the IBM WebSphere documentation for information.

1. Install WebSphere Network Deployment.

2. Run the Profile Creation Wizard to create a Deployment Manager profile. While you are installing, take note of the following information for use during your ATG installation:

    - Deployment manager profile name
    - Deployment manager cell name
    - Administration console port
    - SOAP port

Creating a Cluster

Use the WebSphere Administration Console to create your clusters. The recommended topology for running ATG products on a WebSphere cluster has the following characteristics:

- Includes one Deployment Manager profile and at least one custom profile
- Separates page serving instances and non-page-serving instances into different clusters
- Includes the web servers in the Deployment Manager cell for management

Creating Data Sources

Create your data sources in the WebSphere Administration console; see the documentation for WebSphere and your database solution for information.

Note: The JNDI lookup for your data source must be consistent with the JNDI name configured in your Nucleus-based application. Make sure you set the data source’s scope correctly.

Installing and Configuring Your Web Server

Install your web server and configure it for use with WebSphere; see your web server documentation for information.

Take note of the path used for installing web server configuration files. It is extremely important to copy and run the web server configuration files to your WebSphere servers, so that your application can be targeted to the appropriate web servers and clusters (refer to your WebSphere documentation for more information).

Installing ATG for a WebSphere Cluster

Follow these steps to install the ATG platform to run in a clustered environment:

1. Run the `ATG10.0.1.exe` (Windows) or `ATG10.0.1.bin` (UNIX) file to start the setup program.
2. After you accept the terms of the license agreement, select the installation folder for the ATG software (`C:\ATG\ATG10.0.1` or `/home/ATG/ATG10.0.1`, for example).
3. Select the ATG products you want to install.
4. Select IBM WebSphere – Cluster as your application server.
5. Enter the WebSphere home directory (`C:\WebSphere\AppServer`, for example).
6. Select the Deployment Manager profile and cell.
7. Deploy and install your application (see the WebSphere documentation for information).

Assembling for a WebSphere Cluster

When you invoke the application assembler, use the following flags:

`-standalone`, to assemble the application in standalone mode, so it is not dependent on your ATG installation

`-liveconfig`, to enable the `liveconfig` configuration layer

`-pack`, because the WebSphere application installation wizard does not recognize an exploded EAR file (see Note)

If you are using a named configuration layer, specify that as well (see “Managing Properties Files” in the *ATG Programming Guide* for information on named configuration layers).

Note: It is possible to deploy an exploded EAR through the WAS admin. To do so, in the WAS Deployment Wizard, click the radio button for **Server path** instead of **Local path**, then type in the full path of the EAR directory and submit the form. Note that in order for the WAS deployment wizard to recognize the Server path you provide, the directory must exist on a file system accessible to the server that is serving the WAS admin pages.

Do **not** use the `-server` flag to specify an ATG server configuration.

See the *Assembling Applications* section of the *Developing and Assembling Nucleus-Based Applications* chapter in the *ATG Programming Guide* for more information on assembly.

Session Management in a WebSphere Cluster

When a session is persisted in a database, WebSphere does not correctly invoke the `valueUnbound()` method when that session expires, resulting in memory leaks when running ATG applications. The `/atg/dynamo/servlet/sessiontracking/SessionInvalidationService` component handles this problem by checking the current set of child sessions known to ATG and comparing the last accessed time to the session’s configured timeout, as specified by the application server. If the child session has timed out, it is removed from the list of sessions. When all children of a parent session have been removed, all session-scoped components for that session are cleaned up. The `SessionInvalidationService` runs on a configurable schedule, with a default of every 5 minutes.

Note that this component does not invalidate the session or interfere in any way with the application server’s own cleanup work; it touches only ATG-created items. For even more safety, you can set the `additionalTimeoutMinutes` property, in which case the service waits the specified additional number of minutes above the application server’s configured session timeout before performing the cleanup.

If debugging is turned on, the `SessionInvalidationService` component indicates when it performs a check, and the last accessed time for each child session. The component is defined in the `DafEar.WebSphere` module, which is run automatically on WebSphere.

Configuring Your WebSphere Servers

When you assemble your application, the application assembler includes all of the ATG servers you have configured. This means that you can build your application once, deploy it on each WebSphere application server, and enable the appropriate ATG server on each WebSphere instance simply by changing the value of the `atg.dynamo.server.name` system property when you start up WebSphere.

Do the following for each server.

1. In the WebSphere Administration console, go to **Server** > **Application Servers**.
2. Click the link for the server you want to configure.
3. On the right hand side, go to **Server Infrastructure** > **Java and process management** > **Process Definition** > **Additional Properties** > **Java Virtual Machine**.
4. Enter an initial and maximum heap size; the recommended value is at least 512/512.
5. Return to the **Java Virtual Machine** page.
6. Go to **Custom Properties** > **New**.
7. Create a new system property named `atg.dynamo.server.name`. The value should be the ATG server instance you want to associate with this WebSphere server.
8. If applicable, return to the **Java Virtual Machine** page to enable server mode. In the Generic JVM Arguments field, enter `–server`.
9. Save all changes to the master repository; make sure sync node is enabled.

Deploying Your Application

Use the WebSphere Administration console to deploy your EAR file to a cluster. Each Nucleus-based application needs to be installed as follows:

- If you are deploying your web application to a page-serving cluster, that application should also be deployed to a web server instance.
- If you are deploying the application to a cluster that does not serve pages, but that will run the application, do **not** deploy the application to a web server instance.
- The web server should only route application requests to instances on the web serving instances node, but non-web serving instances will also run the application.

To deploy an application:

1. Using the Administrative Console for the Deployment Manager, install the application.

   If your web application includes a resource reference for your data source, in the WebSphere application installation wizard make sure the reference binding and JNDI name match and are consistent with the name configured in the `JTDatasource` component (excluding the `java:/comp/env` prefix).

2. Regenerate the web server plug-in. In the WebSphere Administration Console, go to **Servers** > **Web servers**. Select the entry corresponding to your web server.

   On IHS with remote web server management enabled, click **Propagate Plug-In**. The plug-in is propagated automatically.

   For all other web servers, click **Update Plug-In**, locate the plug-in on the deployment manager’s file system and transfer it to the web server host; overwrite the existing plug-in.

General Clustering Information

The information in this section applies to all application servers.

Specifying the drpPort Setting

For each ATG server you create, you **must** edit the `Configuration.properties` file in the `<ATG10dir>/home/servers/`*`servername`*`/localconfig/atg/dynamo` directory. Set the `adminPort` property to the listen port of the corresponding application server, and give the `drpPort` property a unique value. For example, for the ATG `procedit` server, you might use these settings:

```
adminPort=7800
drpPort=8851
```

Note that DRP ports are not enabled when you run ATG applications, but the port numbers are still needed to identify scenario server instances. Therefore, you must specify a unique value for the `drpPort` property for the server.

Setting up localconfig and Server Configuration Files

Set up your `localconfig` and server configuration files under `<ATG10dir>/home/servers` and `<ATG10dir>/home/localconfig` to configure the default and server specific behaviors of your Nucleus-based application. These files are included in your EAR when it is generated.

1. Using either the Dynamo Administration UI or the `makeDynamoServer` script, create one ATG server configuration for each application server.
2. Configure the ATG lock manager server to run the ATG `ServerLockManager`. (See Enabling the Repository Cache Lock Managers earlier in this chapter for more information.)
3. Configure the ATG Scenario Manager to run the workflow and process editor servers. There should be exactly one instance of your ATG application running each of these components, and all other instances should be aware of them. (See the *ATG Personalization Programming Guide* for more information.)

Note: The JNDI lookup for your data source must be prefixed with `java:comp/env/`. For example, if your data source has a JNDI name ATGDB, the `JNDIName` property of the `JTDataSource` should be `java:/comp/env/ATGDB`. Set your transaction manager to use your application server's implementation. See the Creating Data Sources section for additional JNDI naming constraints.

Unique Components

The ATG product suite contains several components that must be unique within an ATG server cluster. If you enable and start up more than one instance of these components, errors can result. These unique components are:

- Fulfillment module used by ATG Commerce
- Process editor server used by the Scenario Manager

- Workflow process manager

Fulfillment Module

Only one instance of the ATG Commerce Fulfillment module should run on the system. Only one ATG server instance should be started with the command `startDynamoOnJBOSS -m Fulfillment`. To learn more about the Fulfillment module, see the *ATG Commerce Programming Guide*.

Process Editor Server

A cluster of ATG servers should only contain one process editor server. Make sure you have one process editor server configured and that all other ATG instances are aware of it. See the *ATG Personalization Programming Guide* information about setting up scenario servers.

Because running the global scenario server places an additional burden on your ATG server, this instance should not serve any pages.

Workflow Process Manager

A cluster of ATG servers should always contain exactly one workflow process manager. Make sure only one workflow process manager is configured and that all other ATG instances are aware of it. See the *ATG Personalization Programming Guide* information about setting up a workflow process manager.

Enabling Component Backup

The ATG platform implements a session backup facility that allows you to specify a set of session-scoped or window-scoped Nucleus components and properties that should be backed up after every request. This session backup mechanism saves these components and properties, and restores them when the application server migrates a session to another server.

ATG's component backup works with your application server's persistence facility. To use backup, you must be running your application server in a cluster, and you must enable its in-memory replication form of session persistence for each ATG application (see your application server documentation for information). Note that when you enable in-memory replication for an application, that application must not be deployed on any application server that is not part of a cluster.

To enable ATG's backup, set the `backingUpSessions` property to `true` in the `/atg/dynamo/Configuration.properties` file in the `localconfig` layer.

```
backingUpSessions=true
```

By default, the user's profile and shopping cart (if one exists) are backed up. To back up additional session-scoped components, set the `sessionBackupServerPropertyList` property in the `/atg/dynamo/Configuration.properties` file to a comma-separated list of Nucleus component properties.

Keep in mind when backing up additional information that the more you back up, the more data the app server must save, which could affect performance.

Each component or property specified in `sessionBackupServerPropertyList` must implement `java.io.Serializable` (or `Externalizable`). If a component is listed without any properties, the entire component is backed up.

Synchronizing Server Clocks

Make sure that all server clocks in a cluster are synchronized. Unsynchronized clocks within the cluster can lead to unexpected results.

6 Performance Diagnostics

This chapter includes a checklist that can help you identify performance problems in a systematic way. It also describes tools you can use to look for problem areas and discusses how to analyze the results you get from these tools. This chapter includes the following sections:

Performance Troubleshooting Checklist

As your application nears its launch date, you should test the sites as extensively as possible, using tests that simulate the expected site load as realistically as possible.

If you run into performance problems, you can best identify and correct the source of the problem by taking a systematic approach. The following checklist can help you identify the most common sources of performance problems:

- Have you properly configured memory for your Java Virtual Machines? Have you set your `-Xms` and `-Xmx` arguments the same? Do all ATG heap sizes fall within the limits of physical memory?
- Has one or more servers stopped responding? There could be a number of causes, including a Java deadlock. See Server Hangs.
- Are you seeing many IOExceptions with the message “Too many open files”? You may have a file descriptor leak. See Detecting File Descriptor Leaks.
- At maximum throughput, look at the CPU utilization, database CPU utilization, I/O activity, and paging activity. See Monitoring System Utilization.
- If CPU utilization is low, then you may have an I/O or database bottleneck. See Checking for Disk I/O Bottlenecks, Checking for Network-Limited Problems, and Repository and Database Performance.

- If CPU utilization is high, then the bottleneck is most likely in the application code. Use a performance profiling tool to try to locate bottlenecks in the code. Review your code to make sure it uses good Java programming practices.
- If paging is occurring, adjust the memory allocated to your Java Virtual Machines. See the Swap Space topic in the Paging and Memory Allocation section.
- Look at the I/O and CPU utilization of the database. If utilization is high, database activity is probably slowing down the application. See the Repository and Database Performance chapter.
- Are you receiving page compilation errors? You may not have enough swap space for page compilation.

If your sites develop performance problems, you need to test several paths through your sites to determine the source or sources of the problems. To generate meaningful test results, you need to test sites with loads that achieve maximum throughput.

Performance Testing Strategies

Since your server may be handling requests for different URLs at the same time, there is no way to get throughput statistics on a page-by-page basis. Instead, you may want to run tests with different sequences of URLs to determine how much throughput varies based on what the user is doing on your sites. Some bottlenecks may occur only in certain page sequences. Your ATG installation includes a test utility named URLHammer that you can use to create and run test scripts. See Using URLHammer.

Graduated Testing of Throughput

When you test the performance of your sites, you will get the clearest results if you start with very simple tests. Once you know that individual pages or sequences are performing adequately, you can work toward tests that exercise the full range of functionality on your sites. For example, you might structure your throughput tests as follows:

- a minimal, “hello world” page (tests pipeline/request logging)
- home page (tests a single real page)
- login process
- the 10 most frequently requested pages
- every page

Realistic Testing Strategies

When you load test your sites, be sure to use realistic tests.

- Don’t rely on throughput data from a test script in which 100 separate clients make an identical request at the same moment.
- You will want to test cases where request threads do and do not accept cookies. However, be aware that if you run a performance test in which *every* request does not

accept cookies, your results will reflect a high performance cost from the need to create a session object for each request. You can easily exhaust memory by creating too many sessions.

Locating Performance Bottlenecks

Once you have brought your ATG system up to maximum throughput, you can look at the components of the system to determine which components are limiting factors in performance.

Monitoring System Utilization

Use a program like `top` (on Solaris), the Windows Performance Monitor, or a more sophisticated tool to keep track of information like:

- CPU utilization
- paging activity
- disk I/O utilization
- network I/O utilization

A well-performing site will have high CPU utilization when the site is achieving its maximum throughput and will not be doing any paging. A site with high I/O activity and low CPU utilization has some I/O bottleneck.

Bottlenecks at Low CPU Utilization

If your sites have low CPU utilization when achieving maximum throughput, the bottleneck is likely either:

- database limited (if database output is maxed out); see Checking for Database Bottlenecks
- disk I/O limited (if I/O output is maxed out); see Checking for Disk I/O Bottlenecks
- network I/O limited (if I/O output is maxed out); see Checking for Network-Limited Problems
- database or I/O activity in a synchronized method (if database or I/O output is not maxed out); see System Resource Bottlenecks

If your site is in this situation, CPU profiling tools are not that useful. Thread dumps taken while the system is under load can give you better information. If you take a few of these, you can get a quick idea of which parts of your application are the slowest. That may help you direct your efforts to the right part of your application. You should be able to tell, for example, whether threads are waiting for a response from the database, a write to the client, or a read from a local file system. If many threads are waiting for the same resource, this is an indication of a potential bottleneck on that resource. Here is some information on what to do about resource bottlenecks for various resources:

Checking for Database Bottlenecks

If your site has low CPU utilization at maximum throughput, check whether the database is limiting performance.

- Get a JVM thread dump and examine it to see if there are many threads waiting for a response from the database.
- Check the CPU utilization and disk I/O utilization of your database server.
- Check the network bandwidth between the ATG server and the database server.

For more information about improving database performance with ATG, see the Repository and Database Performance chapter.

Checking for Disk I/O Bottlenecks

Make sure that your JVM really is waiting for file I/O, not paging activity. Check for paging with your operating system's monitoring tools.

If the source of slow performance is file I/O, it will show up in JVM thread dumps. The cause could be either some application-specific code that you have, or else the file I/O that ATG does itself.

Checking for Network-Limited Problems

One way to identify network-limited performance problems is by getting your JVM to dump out stack traces while your system is under load. You can tell if your system is network limited because your thread dump will show lots of threads waiting in socket reads or writes.

Some ways to address network-limited problems include:

- Reduce the size of your HTML files by limiting comments and white space or redesigning the content of especially large pages.
- Increase the number of request handling threads. This won't improve the latency experienced by a user who requests a large file, but it will improve total throughput.
- Get a faster network connection.
- Locate and correct network bottlenecks.

Bottlenecks at High CPU Utilization

If your site CPU utilization is close to 100%, you can use a Java profiler tool like JProfiler or JProbe Profiler to help determine slow points of your code.

In some instances, profilers cannot handle large sites running under load. If so, another way to identify deadlocks and bottlenecks is to get your JVM to dump out stack traces while your system is under load. If you examine 5 or 10 of these stack traces, you can start to see a pattern and find places in your site that are consuming CPU resources or causing deadlocks.

An alternative to stack dumps is the HPROF utility provided with the JDK. See Oracle's Java documentation for information on this utility.

Thread Context Switching Problems

Check how many simultaneous requests are typically being handled when you have a large number of clients trying to access your application. Thread dumps can be useful to see where these threads are waiting. If there are too many threads waiting, your site's performance may be impaired by thread context switching. You might see throughput decrease as load increases if your server were spending too much time context-switching between requests. Check the percentage of System CPU time consumed by your JVM. If this is more than 10% to 20%, this is potentially a problem. Thread context switching also depends in part on how your JVM schedules threads with different priorities.

You can also reduce overhead from thread context switching by making sure you have at least one CPU for each process involved in handling the majority of requests: one CPU for your HTTP server, one for ATG, one for the database server.

You might see throughput go down as load increases in cases where all of your request handler threads were busy waiting for some resource at the same time. For example, you might have one page on your site that makes a very long-running database query. If you increase the number of clients well beyond 40, you might see all 40 threads waiting for the response to this query. At this point, your throughput will go down because your CPU is idle. You should either speed up the slow requests (perhaps by adding caching of these queries) or increase the number of request threads to increase the parallelism. Of course, at some point, the database may become the bottleneck of your site (which is likely before you have 40 simultaneous queries running).

Context switching can also occur when you have a network protocol which synchronizes too often (such as sending a request and waiting for a response).

Typically, these context switches can be overcome by increasing the parallelism in your site. If there are just too many of these synchronization points, though, this won't work. For example, if you have 40 synchronous RPC calls for each HTTP request, you'd need to context switch processes 80 times for each request if you handled one request at a time. If you handled 2 requests at a time, you'd cut the number of context switches in half. This is in addition to the number of handlers that you'd need to hide any I/O or database activity so the number can add up fast.

System Resource Bottlenecks

If your site has not maxed out either CPU utilization, database server utilization, or I/O subsystem, the problem may result from synchronized access to one of your system's resources (such as disk, network, database, etc.). This situation occurs when you access this resource from within a synchronized method in Java. All other requests wait for this monitor lock while you do the I/O, thus wasting both CPU and I/O resources. The only ways around this problem are to recode the Java (the right solution) or add more ATG instances (the wrong solution).

The easiest way to find these problems is to test your site when it is serving pages under load and get a JVM thread dump. By examining the thread dump, you may see one thread waiting for a response from the OS (database or I/O) and a set of other threads waiting on a monitor lock that this other thread has.

Lower Thread Priorities

If you have a rarely used feature that uses a lot of CPU resources, you can lower the priority of the thread that handles requests for that feature. Use the `setPriority()` method of `java.lang.Thread` to temporarily lower the thread priority. This will result in higher latency for users of that expensive feature, but prevents that feature from hurting performance of other users.

TCP Wait Problem on Solaris

In some testing situations involving a very large number of requests from a single client on the Solaris platform, you may see a dramatic and periodic decline in throughput. You may be able to correct this by modifying the `tcp_close_wait_interval` setting in the `/dev/tcp` module. You can do this in two different ways:

- Start `ndd`, access the `/dev/tcp` module, and change the value of `tcp_close_wait_interval` to 60000 (60 seconds).
- Edit the `/etc/init.d/inetinit` file and include the following line:

  ```
  ndd -set /dev/tcp tcp_close_wait_interval 60000
  ```

Server Hangs

If one or more servers on your site stops responding unaccountably after running under load for a certain period of time, there are a few possible causes:

- HTTP servers not sending requests to your application.
- A Java deadlock.
- Some resource that your application depends on is itself hung (such as the database or some service with which the application communicates via sockets). For example, if a single client opens up hundreds of connections to request pages and then stops reading the response data, this could lock up a server without any real failure of any ATG components.
- You may also have consumed all of the memory in your JVM. If this happens, you'll usually see `OutOfMemory` errors in your console right before the server hangs. This may appear as a hang because the server will do a garbage collection to reclaim a few bytes, run a few lines of code, then walk through the heap again trying to find another few bytes to reclaim.
- An infinite loop in some code.

Here are some steps you can take to attempt to identify the cause of the server hang.

- Check the CPU utilization of the machine and particularly the Java process running your ATG application. If CPU utilization is 100%, it is either an `OutOfMemory` problem or a CPU burning thread.
- Check the server logs to see if any errors right before the hang indicate why the server has failed. You might see a "server not responding" message or an `OutOfMemory` error.
- Get a thread dump from your Java VM. A thread dump can help you recognize all of these problems.

If all threads are waiting in system calls such as socket read/write, then they are waiting for a resource to respond (for instance, the database or the network). You should look to this resource for answers. If the resource is a database, try using a third party database tool to make a query. It is possible that the tables used by your ATG application are locked by some other operation so they will wait until that operation has completed.

Paging and Memory Allocation

If you see any paging activity, increase system memory or decrease the size of the JVMs. Be aware that decreasing heap sizes may increase the overhead of garbage collection. Each time a full garbage collection is performed, all of the memory needs to be scanned for garbage. Garbage collections occur more frequently with smaller heaps, which could waste CPU time.

You can check the size of your JVM heaps or cause garbage collection with the ATG VMSystem component at:

```
http://hostname:port/dyn/admin/nucleus/VMSystem/
```

Garbage Collection

Set your JVM to monitor how much time is spent doing garbage collection. You can do this by adding the `-verbose:gc` parameter to the JAVA_ARGS passed to the JVM when ATG starts up. The `-verbose:gc` option causes the JVM to output a message each time garbage collection is performed, including:

- how much memory was reclaimed
- the amount of free memory
- the total heap size
- how much time the garbage collection operation took

If you see your garbage collections happening too often or occupying a significant percentage of your CPU, you should either increase the Java heap size arguments or look for places in your application that are allocating memory unnecessarily.

If the garbage collection takes a very long time to complete, you may have configured your heap size to be too large for the amount of memory your system has. If your system is spending more than 20% of its CPU time on garbage collection, you have a significant performance problem that must be corrected. Use your OS monitoring tools to see if you are paging and check the process size of all of the processes running on your system. Compare this with the physical memory of your machine.

If your heap is very large, your garbage collections may occur very infrequently but may take a long time (30 seconds or more) to complete. This is a structural limitation of Java that is difficult to work around. When a full garbage collection occurs, it typically acquires the heap lock, which prevents all requests from being served during this time interval. You can potentially reduce the time of these garbage collections by forcing them to occur more frequently.

If your garbage collections take a significant percentage of your overall CPU time, you may have places in your code that allocate memory inefficiently. It is a good idea to reuse objects where possible, rather than creating them over and over again. You can use a memory profiler to determine where and how much memory is allocated by which places of the code. Only the allocation side of the garbage shows up in the stack traces and profiling. You have to factor in the time spent reclaiming garbage as well. You can also use the Performance Monitor to trace the memory allocation of various operations in your system. See Performance Monitor in the Monitoring Site Performance chapter.

Memory Leaks

When the Java VM runs low on memory, you should see two behaviors:

- very slow performance, as garbage collections occur more frequently and absorb a greater share of CPU time
- occasional `OutOfMemory` errors.

To confirm the presence of a memory leak, add `-verbose:gc` to your JAVA_ARGS and monitor the number of sessions on your site (see the Garbage Collection section for details). If you see free memory decrease over time as your site has a constant number of sessions, you may have a memory leak. Before deciding that you have a memory leak, make sure you have given the system enough time to fill all caches and reach a stable state after startup.

Memory leaks in Java are caused by data structures that hold onto objects that are no longer needed. This is often due to a Collection (such as a Vector or Hashtable) that is not coded correctly. For example, if you store objects in a Hashtable using a session ID as a key, but you do not remove these objects when the session expires, this Hashtable will grow without bounds.

You can use memory profilers to help find these errors. Another way to detect when a Hashtable or Vector is growing without bounds is to use a modified version of the standard Hashtable and Vector that is instrumented to print a message each time the 10000th, 20000th, etc. element is added. Of course, if you use a different Collection class, this will not find that problem.

One frequent cause of Java memory leaks is the use of an `addXXXListener()` method without a corresponding `removeXXXListener()` method. Review your code to make sure you haven't made this mistake.

Swap Space

In order for ATG to fork a `javac` compiler to compile a JHTML page, it requires two times the current process size in swap space for a short period of time until it executes the new process. If you receive an error message like this:

```
/atg/dynamo/servlet/pagecompile/PageCompileServlet
atg.servlet.pagecompile.PageCompileResources->
pageCompileServletErrorCompiling :
Error compiling page: <path of page> :
Unable to execute the command '<page compile command>'
Make sure that you have the 'bin' directory for your JDK in your PATH
variable before starting ATG and that you have enough swap space.
```

then you probably do not have enough swap space for page compilation. Increase your swap space.

Detecting File Descriptor Leaks

It is important to ensure that files that are opened always get closed. Failing to close files can result in file descriptor leaks. You can detect a file descriptor leak in two different ways:

- You may notice a lot of `IOExceptions` with the message “Too many open files.”
- During load testing, you periodically run a profiling script, such as `lsof` (on UNIX), and you notice that the list of file descriptors grows continually.

File descriptor leaks can also lead to a variety of failures on attempts to open properties files, sockets, etc. If your error log contains a lot of chaotic-looking error messages, the presence of a file descriptor leak is one thing to check.

Using URLHammer

The URLHammer program is a Java utility. URLHammer makes repeated page requests, allowing you to simulate the effects of load on your ATG application. The utility detects and reports HTTP errors, but performs no validation of the HTTP response itself. URLHammer supports HTTP cookies. You can use it to submit forms by playing back scripts (see Using the Recording Servlet). URLHammer is run from the DOS or UNIX command line. It runs in a separate JVM from ATG. For the best results, we recommend running URLHammer on a separate machine from the server you are testing.

To run the URLHammer program:

1. Set your CLASSPATH to include the directory `<ATG10dir>/DAS/lib/classes.jar`.
2. Run the following command:

   ```
   java atg.core.net.URLHammer [arguments]
   ```

For example:

```
java atg.core.net.URLHammer http://examplehost:8840/ 5 10 -cookies
```

This creates five different threads, each of which represents a separate session that requests the specified URL 10 times (50 requests total).

You can configure URLHammer using several command line arguments that are described below; you can also use the `-usage` argument to get the current list of arguments. The `-cookies` argument makes URLHammer parse the `Set-cookie` headers and return cookies that it receives in all subsequent requests. If you don’t use the `-cookies` argument, then ATG creates new sessions for each request. Each thread has its own set of cookies. Thus, the above example creates 5 sessions and executes 10 requests in each.

Command Line Arguments

URLHammer takes a number of command line arguments so that you can implement your tests in the manner that best fits your site. Use the following syntax:

```
java atg.core.net.URLHammer URL | script_pathname threads iterations
[optional arguments]
```

The following URLHammer arguments are required:

Required Arguments	Description
URL or *script_pathname*	The URL to use in each request. The URL must begin with `http://`. (**Note:** https is not supported.) If you use the `-script` argument, then instead of a URL, specify the pathname of the script to execute. See The -script Argument.
threads	Number of independent thread connections to create to the HTTP server. Use a value from 1 to 20. All threads run concurrently.
iterations	Number of requests to issue on each thread. If the `-script` argument is used, this represents instead the number of times each thread executes the entire script.

The following URLHammer arguments are optional:

Optional Arguments	Description
`-addCookie` *name=value*	Enables you to set a cookie. For example: `-addCookie FOO=Zippy`
`-addHeader` *name=value*	Enables you to define a header. You can define multiple headers; for example: `-addHeader LOGIN=Zappa -addHeader PASS=nan00k`
`-cookies`	Returns `Set-cookie` headers sent by the server. Note: `path=` and `expires=` are not processed by URLHammer.
`-htmlStats` *HTML file*	Output statistics to the specified HTML file. This argument gives detailed statistics about the amount of time consumed by each individual URL you requested. It also gives summary statistics about the number of errors encountered. By default, URLHammer outputs these statistics to the console.
`-maxRequests`	Limits the number of redirects that can be generated.

`-nopause`	Use only with the `-script` argument. Ignores pause information in script files by default. When using the Recording Servlet, the time between the server’s receipt of one page request and the server’s receipt of the next request is recorded in the script file (in milliseconds). Each URLHammer thread sleeps for this number of milliseconds before requesting the URL. If you use the `-nopause` argument, URLHammer instead requests each subsequent URL as soon as the previous output is received.
`-password`	Use only with the `-user` argument. Supplies a user password if needed to log in to any pages.

`-pause`	Use only with the `-script` argument. Pause for the specified time between each request (number of milliseconds). For example, the following argument causes URLHammer to pause 1 second between each request:
	`-pause 1000`
	If you use a negative value, then URLHammer pauses for a random amount of time, not to exceed the absolute value of the value you use. For example, the following argument causes URLHammer to pause a random amount between 0 and 550 milliseconds between each request:
	`-pause –550`

`-randomStop`	Simulates the browser’s stop button by randomly closing the connection to the server for 20% of the requests.
`-recordAll`	Outputs statistics for each request. Use this argument with the `-htmlStats` argument and the HTML file will contain the statistics broken down for each request, as well as in summary form. It also keeps track of which requests had errors and prints `(error)` next to the time for that request.
`-runningStats`	Prints information periodically for each thread. This allows you to get an idea of how long runs are proceeding.
`-script`	Instead of making a request to a single URL, each thread instead executes a script of user browser actions. See The -script Argument.
`-server` *`name`*`:`*`port-number`*	Name of the server and the port number to use if you are using the `-script` argument. If you do not specify a server, `localhost:80` is used as the default.
`-stop` *`<n>`*	Simulates the browser’s stop button by closing the connection to the server for *<n>*% of the requests. This argument is useful to make sure that your site is robust with respect to aborted requests.

`-substitute`	Use only with the `-script` argument. Performs keyword substitution in your script file. This facility allows you to generate more flexible form processing scripts. You can place keywords `__RANDOM__`, `__COUNTER__`, and `__TIME__` into your script file's URLs and POST data sections. (Note that these keywords are preceded and followed by two underscore characters.) Before each request, URLHammer substitutes these keywords with a random string, a continually incremented counter, or the current time in milliseconds. You can use this argument, for example, to generate unique login IDs when load testing login forms.
`-user`	Use only with the `-password` argument. Supplies a username if needed to log in to any pages.
`-verbose`	Dumps the complete output of the request (including request headers). This argument is very valuable when testing a new script or the first time you execute a command, so that you can inspect the output generated.

URLHammer Examples

The following examples use UNIX syntax. Adjust the syntax accordingly for Windows. We also presume that your `CLASSPATH` includes `$DYNAMO_HOME/lib/classes.jar`.

Checking Availability of ATG

Suppose you want to see whether your ATG application is responding. A single request on a single thread, using a very simple page, would be sufficient for this test:

```
java atg.core.net.URLHammer http://hostname:8080/index.jsp 1 1
```

If your application is responding, you should see output like the following (the times will vary):

```
Time = 521 ms (1.91 requests/s; average latency = 521 ms)
0 errors out of 1 request
```

The time output reports the total elapsed time in milliseconds, the number of requests per second, and the average time per request, in milliseconds.

Generating a Typical Load

Using multiple concurrent threads, each making repeated requests, will generate a sustained load on the ATG server:

```
java atg.core.net.URLHammer http://hostname:8080/test.jsp 10 25
```

In this example, 10 threads are used, each making 25 requests, for a total of 250 requests, each of which uses its own session.

Playing Back a Script

The previous examples generate a number of simultaneous requests for the same page. For a more realistic usage scenario, you can use URLHammer to run a script of more complex user behavior. A script file can be as simple as a list of relative URIs (one per line). See Recording a Script for a simple way to construct a script, and Editing a Script for details on the syntax and semantics. The following command plays back the script `myscript.txt` one time, using one thread, making requests from the default ATG server port:

```
java atg.core.net.URLHammer myscript.txt 1 1 -script -server
examplehost:8080
```

The -script Argument

The `-script` argument treats the URL argument as the name of a script file on the local system. This script file can contain any of the following:

- URLs
- URLs with POST data
- URLs with POST data and session ID arguments

You can write your own script files, or you can use ATG's Recording Servlet, which records script files that replay a previously recorded set of user actions. See Using the Recording Servlet in the Monitoring Site Performance chapter.

Script files are line-oriented ASCII text files. Each line can be in one of the following formats:

```
#include another_script_file

URL

URL time_in_milliseconds

URL time_in_milliseconds #_lines_of_post_data
post_data
post_data
post_data
...

URL time_in_millis #_lines_of_post_data  session_id
post_data
post_data
post_data
...
```

If a line specifies a number of lines of POST data, URLHammer reads that number of lines and passes them as URL-encoded POST data to the specified URL. Typically, lines of this form are generated by the Recording Servlet.

Note that the URLs in a script file need not contain the `http://`*`hostname`*`:`*`port`* prefix, since the full URL can be constructed using the host and port number specified by the `-server` command line argument. This allows you to reuse the same script to test different servers.

Recording a Script

You can use the ATG Recording Servlet facility as an aid in constructing a test script. This is particularly helpful in tests of form submission (such as requests with the POST method) because the script must supply the data for the form. Follow these steps to record a test script:

1. Open the `/atg/dynamo/servlet/pipeline/RecordingServlet` component in the ACC.
2. If the `RecordingServlet` component is not running, start it by clicking the Start button.
3. Change the live value of the `recording` property to `true`.
4. Perform the actions you wish to record (for example, page requests and submitting forms).
5. Change the live value of the `recording` property to `false`.
6. Copy the `<ATG10dir>/home/logs/record.log` file to another filename to save its contents.

You can also use the Recording Servlet with the Dynamo Administration UI:

1. Browse the Recording Servlet in the Dynamo Administration UI:

   ```
   http://hostname:port/dyn/admin/nucleus/atg/dynamo/servlet/pipeline/
   RecordingServlet
   ```

2. Click the name of the `recording` property.
3. Set the value to `true` and click the `Change Value` button.
4. Perform the actions you wish to record (for example, page requests and submitting forms).
5. Return to the Recording Servlet page in the Dynamo Administration UI.
6. Click the name of the `recording` property.
7. Set the `recording` value to `false` and click the **Change Value** button.
8. Copy the `<ATG10dir>/home/logs/record.log` file to another filename to save its contents.

See also Using the Recording Servlet in the Monitoring Site Performance chapter.

Editing a Script

A request in a script file is specified using this syntax:

```
Relative_URI [ Delay_ms [ POST_lines [ Session_ID ] ] ]
```

where:

- `Relative_URI` is the relative URI of the file to request, with optional parameters
- `Delay_ms` is the number of milliseconds to pause
- `POST_lines` specifies the number of following lines to use as POST data
- `Session_ID` designates an ATG session ID

The URIs in a recorded script must be relative to the document root. Note also that when the `-cookies` option is used, all of the session IDs in a script are replaced by the current session ID for the given thread; each thread will have a new unique session created for it.

Comments in Scripts

A line that begins with the # character is considered a comment and will be ignored (with the exception of lines that begin with `#include`; see next section). You can add comments to your scripts to document the purpose, author, usage, etc.

Including Scripts within Scripts

A line that begins with the `#include` keyword includes a specified script within the current script. For example:

```
#include subfile.txt
```

adds the contents of the script `subfile.txt` to the current script at that position. This is especially useful for simplifying a long script into a hierarchy of easy-to-understand parts.

URLHammer Source Files

ATG includes the source for URLHammer, together with source for implementation classes, in:

```
<ATG10dir>/DAS/src/Java/atg/core/net/
```

You may want to modify or extend URLHammer for your own testing purposes. However, ATG does not guarantee backward compatibility in future releases of URLHammer. If you make modifications to the code, you should change the class and package names to avoid potential conflicts with future versions we may release.

7 Monitoring Site Performance

ATG includes a variety of diagnostic and administrative tools to help you keep your site up and running smoothly. This chapter covers the following topics:

Performance Monitor

ATG's Performance Monitor component provides a tool you can use to monitor the performance of regions of your code. To use the Performance Monitor:

- Instrument your Java code with static methods that enable the Performance Monitor to gather information about performance (see Adding PerformanceMonitor Methods to your Code).
- View the Performance Monitor page in the Dynamo Administration UI to inspect information gathered (see Viewing Performance Monitor Data).

The Performance Monitor can run in different modes. In normal (default) mode it causes negligible overhead, but allows you to globally turn on one or more monitoring options which give more diagnostic information. These monitoring options would typically be used during load testing but are not suitable for running on a live site under heavy load. See Performance Monitor Modes.

Adding PerformanceMonitor Methods to your Code

To enable the Performance Monitor to monitor a section of your Java code:

1. Import the `atg.service.perfmonitor.*` package.
2. Declare an `opName` parameter to label the section of the code. This parameter is displayed in the Performance Monitor page under the **Operation** heading.
3. (Optional) Declare a parameter name if you want to gather data on individual executions of an operation.

4. Call the `startOperation` method at the beginning of the operation whose performance you want to be able to measure.

5. Call the `endOperation` method at the end of the operation whose performance you want to be able to measure.

6. Optionally, call the `cancelOperation` method if an exception occurs. This causes the results of the current execution to be ignored.

For details about the Performance Monitor's `startOperation`, `endOperation`, and `cancelOperation` methods, see Methods for Storing Performance Data.

For example:

```
String opName = "render jsp";
String parameter = "foo.jsp";
boolean exception = false;
PerformanceMonitor.startOperation(opName, parameter);
try {
  ... code to actually render foo.jsp
} catch (Exception e) {
  PerformanceMonitor.cancelOperation(opName, parameter);
  exception = true;
} finally {
  if (! exception)
    PerformanceMonitor.endOperation(opName, parameter);
}
```

These methods can be nested with different or the same `opNames`. For example:

```
private final String RENDER_JSP = "Render JSP page";
private final String EXECUTE_SQL = "Execute SQL Query";
private String mPageName = "page.jsp";
private String mSQLQuery = "select * from table";

PerformanceMonitor.startOperation(RENDER_JSP, mPageName);
... source code to start render
  PerformanceMonitor.startOperation(EXECUTE_SQL, mSQLQuery);
  ... source code to read from table 1 in database
    PerformanceMonitor.startOperation(EXECUTE_SQL);
    ... source code to read from database
    PerformanceMonitor.endOperation(EXECUTE_SQL);
  ... more source code to read from table 1 in database
  PerformanceMonitor.endOperation(EXECUTE_SQL, mSQLQuery);
... more source code to finish render
PerformanceMonitor.endOperation(RENDER_JSP, mPageName);
```

Note that the calls to `startOperation` are nested within other calls to `startOperation`. You must place the `endOperation` and `cancelOperation` calls in the code in opposite order that the `startOperation` calls were placed. If this requirement is not followed, then the `endOperation` or `cancelOperation` call throws a `PerfStackMismatchException`. This exception tells you that the calls to `endOperation` are not being matched up. Either they were not called in the correct order or the arguments were not exactly the same as those that were passed into the methods.

To ensure that `endOperation` is always called, wrap the Performance Monitor methods in a `try ... finally` block, as in this example:

```
boolean exception = false;
try {
  PerformanceMonitor.startOperation(OP_NAME);
  performOperation (pParameter);
} catch (Exception e) {
  PerformanceMonitor.cancelOperation(OP_NAME);
  exception = true;
} finally {
  try {
    if (!exception)
      PerformanceMonitor.endOperation(OP_NAME);
  } catch (PerfStackMismatchException e) {
    System.out.println(e);
  }
}
```

Performance Monitor Modes

The Performance Monitor code can run in one of four modes:

- **DISABLED**. When the Performance Monitor is disabled, its diagnostic methods immediately return without doing any additional work.
- **NORMAL**. In this mode, the Performance Monitor keeps track only of the current stack of operations. This mode is useful in identifying the location in the code of hung or active threads.
- **TIME**. In this mode, in addition to the current operation stack, the Performance Monitor maintains dictionaries for each operation. These dictionaries store the number of times each operation has been performed, and the minimum, maximum and average time to process that operation.
- TIME mode is not meant to be used on a live system for an extended period of time. This mode is for gathering data on the amount of time spent in various parts of the code.
- **MEMORY**. In this mode, the Performance Monitor maintains the information specified for NORMAL and TIME mode. In addition, the Performance Monitor maintains dictionaries that store the number of times each operation has been performed, and the minimum, maximum and average amount of memory required to process that

operation. These statistics are estimates and do not take into account asynchronous processing activity that may be occurring. Do not rely on data from only one or two samples, since the Performance Monitor may generate anomalous data that can be ignored.

MEMORY mode causes all requests to the server to be serialized and could possibly cause deadlock. This mode is provided for diagnostics during development only and is not suitable for use on a live system.

Setting the Mode

Set the Performance Monitor's operating mode at the Performance Monitor Configuration page of the Dynamo Administration UI:

```
http://hostname:port/dyn/admin/atg/dynamo/admin/en/
performance-monitor-config.jhtml
```

Click the radio button for the mode you want, and then click the **Change Mode** button.

You can also set the Performance Monitor's operating mode by setting the `mode` property of the component at `/atg/dynamo/service/PerformanceMonitor`. The value of the `mode` property is an `int` corresponding to the mode:

mode	**int value**
disabled	0 (default)
normal	1
time	2
memory	3

Viewing Performance Monitor Data

You can view the information collected by the Performance Monitor on the Performance Monitor's page of the Dynamo Administration UI at:

```
http://hostname:port/dyn/admin/atg/dynamo/admin/en/
performance-monitor.jhtml
```

This page displays any information recorded by the Performance Monitor. Under the **Threads** heading, the Performance Monitor page displays the operation stack of the current thread.

If you have configured the Performance Monitor to run in TIME mode, then the Performance Monitor page displays under the **Performance Data** heading a Time Performance Data table with a list of operations that have been recorded (such as `Invoke Servlet`, `Compile Page`, `Service Request`, etc.) along with the number of times the operation was executed and the minimum, maximum, average, and total time for each.

For example, the Time Performance Data table might look like this:

Operation	Number of Executions	Average Execution Time (msec)	Minimum Execution Time (msec)	Maximum Execution Time (msec)	Total Execution Time (msec)
Handle HTTP Request	1	223	223	223	223
Invoke Servlet	4	8	0	19	35
Invoke Form Handler	1	108	108	108	108
Compile Page	1	3	3	3	3
Service Request	1	123	123	123	123

The name of each operation is a link to another administration page that provides the detailed parameterized information, if any (for example, for each URL, the number of times requested, the minimum, maximum, and average times).

If you have configured the Performance Monitor to run in MEMORY mode, then the Performance Monitor page displays under the **Performance Data** heading Time and Memory Performance Data tables that includes all the TIME mode information described above, and in addition displays the minimum, maximum, average, and total *memory* used by each operation.

Instrumented ATG Classes

Several common ATG operations have already been instrumented with Performance Monitor `startOperation` and `endOperation` methods. By default, this includes all scheduled jobs handled by the Dynamo Scheduler. These operations appear grouped together under the line **Scheduled Jobs** in the Performance Monitor page. Clicking on this link lets you drill down and see the statistics for each job separately. If you don't want performance monitoring of scheduled jobs, you can set the Scheduler's `performanceMonitorEnabled` property to `false` to disable this behavior. See the *ATG Programming Guide* for more information about the Scheduler service.

In addition, ATG's instrumented methods include:

Class Name	Method	Operation Name
`atg.targeting.TargetingArray`	`getTargetArray()`	Perform Targeting
`atg.servlet.pipeline.HeadPipelineServlet`	`service()`	Service Request
`atg.servlet.pagecompile.SubServlet`	`serviceByName()`	Invoke Servlet
`atg.servlet.pagecompile.PageSubServlet`	`serviceServlet()`	Invoke Servlet

`atg.servlet.pagecompile.` `PageCompileServlet`	`service()`	Render Page

`atg.service.resourcepool.`
`MonitoredStatement`
	`executeQuery()`
	`executeUpdate()`
		Execute Query
		Execute Update

`atg.service.resourcepool.`
`MonitoredPreparedStatement`
	`executeQuery()`
	`executeUpdate()`
		Execute Query
		Execute Update

`atg.server.http.HttpConnection`	`handleRequest()`	Handle HTTP Request
`atg.nucleus.NucleusNameResolver`	`createFromName()`	Create Component
`atg.droplet.DropletEventServlet`	`sendEvents()`	Invoke Form Handler
`atg.service.pipeline.PipelineManager`	`runProcess()`	Run Pipeline Chain
`atg.service.pipeline.PipelineLink`	`runProcess()`	Run Pipeline Processor
`atg.adapter.gsa.GSARepository`	`createNewItem()`	GSA createItem
`atg.adapter.gsa.GSAItemDescriptor`	`getPersistentItem()`	GSA Uncached getItem

Performance Monitor API

The main class for the Performance Monitor is `atg.service.perfmonitor.PerformanceMonitor`. This class contains all the static methods for interacting with the Performance Monitor. In addition, it stores the data structures that contain the performance data. The Performance Monitor's methods have the following functions:

- Methods for Controlling the Performance Monitor
- Methods for Storing Performance Data
- Methods for Accessing Stack Data
- Methods for Accessing Performance Data
- Exception Summary

The `PerformanceMonitor` component contains two primary data structures. One stores the runtime stack data for all registered threads. The other stores the performance data for operations and parameterized operations on those registered threads.

Runtime Stack Data Structure
This structure is a `Hashtable` where the key is a registered thread and the element is a `java.util.Stack` of `atg.service.perfmonitor.PerformanceStackData` objects. This data is what is recorded and tracked in `NORMAL` mode. When a stack becomes empty, then all the performance operations have completed in that thread. This data structure is used in all modes except for `DISABLED`.

Performance Data Structure
This data structure stores all the time and memory performance related data for operations and parameterized operations. It is only used when the mode for the

Performance Monitor is set to TIME or MEMORY. The structure is a Hashtable where the key is an operation name and the element is a PerformanceHashtable. The PerformanceHashtable is a subclass of Hashtable. In addition to providing the services of a Hashtable, it also stores the totals for all the parameterized operations contained in the Hashtable in an atg.service.perfmonitor.PerformanceData object. The Hashtable in the superclass of this object contains the parameterized operation name in the key and a PerformanceData object as the element.

There are also two data structures for holding pools of PerformanceStackData and PerformanceData objects. These exist to avoid allocation and improve performance. When startOperation is called, a new PerformanceStackData object is retrieved from the pool, populated and pushed on the stack. When endOperation is called, the top element in the stack is compared for mismatch and then popped off the stack, assuming there was no mismatch. At this time, the corresponding PerformanceData object for the operation in the PerformanceStackData object which is stored in the performance data structure is updated with number of times executed and total execution time (min and max will also be updated if the most current execution requires it). In addition, the global PerformanceData object for the operation is updated. If endOperation was called with no parameterized data, then only the global PerformanceData object for the operation is updated. If the PerformanceData object for the operation or parameterized data does not exist, then a new PerformanceHashtable will be created and PerformanceData object will be retrieved from the pool and inserted.

Methods for Controlling the Performance Monitor

You can control the Performance Monitor programmatically using the methods listed in this section. Most often, however, you will configure the Performance Monitor using the Performance Monitor Configuration page in the Dynamo Administration UI (http://*hostname*:*port*/dyn/admin/atg/dynamo/admin/en/performance-monitor-config.jhtml) or through the ACC.

```
public int getMode();
```
Returns the mode that the Performance Monitor is running in. The return value is an int that refers to one of DISABLED, NORMAL, TIME, or MEMORY.

```
public void setMode(int pMode);
```
Allows a user to dynamically set the mode of the Performance Monitor. The mode is normally set in the Performance Monitor's properties file, but can be changed during runtime using the ACC.

```
public void resetPerformanceData();
```
Resets all the performance data back to 0. This means that the TIME mode and MEMORY mode minimum, maximum, and total statistics will be reset to 0 for all operations and parameterized operations.

Methods for Storing Performance Data

The startOperation and endOperation methods designate the start and end of an operation. These methods need to bracket the code that performs the designated function.

```
public static final void PerformanceMonitor.startOperation(String pOpName);

public static final void PerformanceMonitor.startOperation(String pOpName, String
pParameter);
```

The startOperation method tells Performance Monitor that a new operation is starting. The pOpName parameter is the name of the operation. This parameter should be short and as descriptive as possible. The next parameter, pParameter, is optional data that gives the Performance Monitor more detailed information on exactly what object it is performing the given operation on. The parameterized version of this method records data for the operation on the given parameter and the global operation. The non-parameterized version of this method records performance data to the operational level only.

```
public static final void PerformanceMonitor.endOperation(String pOpName)
     throws PerfStackMismatchException;

public static final void PerformanceMonitor.endOperation(String pOpName, String pP
arameter)
     throws PerfStackMismatchException;

public static final void PerformanceMonitor.endOperation();
```

The endOperation method tells Performance Monitor that a previously started operation has come to completion. The pOpName parameter must be exactly the same as the pOpName parameter that was passed into the corresponding startOperation method. The pParameter is optional data which gives the Performance Monitor more detailed information on the object it completed the operation on. The call to endOperation must have exactly the same parameters that the call to startOperation did. Otherwise, a PerfStackMismatchException (an extension of RuntimeException) is thrown.

You can also call endOperation without any arguments to mark the end of the most recent operation for which monitoring has started, but not yet ended. In this case, there is no need to supply it with the same arguments that were passed at the start of the operation. Accordingly, it will never throw an exception.

The cancelOperation method cancels an operation and discards any performance statistics.

```
public static final void PerformanceMonitor.cancelOperation(String pOpName)
     throws PerfStackMismatchException;

public static final void PerformanceMonitor.cancelOperation(String pOpName, String
 pParameter)
     throws PerfStackMismatchException;
```

The cancelOperation method tells Performance Monitor that a previously started operation should be cancelled. Canceling an operation means that statistics from this operation execution are discarded. The pOpName parameter must be exactly the same as the pOpName parameter that was passed into the corresponding startOperation method. The pParameter is optional data which gives the Performance Monitor more detailed information on the object on which it completed the operation. The call to cancelOperation must have exactly the same parameters that the call to startOperation did. Otherwise, a PerfStackMismatchException is thrown.

The isEnabled method indicates whether the Performance Monitor is enabled or not.

```
public static final boolean PerformanceMonitor.isEnabled();
```

Returns a boolean that specifies whether the Performance Monitor is enabled or not.

Methods for Accessing Stack Data

The stack data contains the runtime location of all the threads currently registered in the Performance Monitor. This data is stored in objects of type `PerformanceStackData`. The `PerformanceStackData` object is contained in a `java.util.Stack` object. The `PerformanceStackData` object alone is not useful; it becomes useful when it is placed inside the context of a `java.util.Stack`. The `PerformanceStackData` has the following methods you can use:

```
public String getOperation();
```
Returns the operation name within the `PerformanceStackData` object.

```
public String getParameter();
```
Returns the parameter operation name within the `PerformanceStackData` object.

```
public long getStartTime();
```
Returns the start time of the operation as the number of milliseconds since Jan 1, 1970. This method is used internally by the Performance Monitor and is not very useful outside of it, but it is provided.

Methods for Accessing Performance Data

The performance data is stored in read-only properties in objects of type `PerformanceData`. This object is a JavaBean that contains the following data:

Property	**Description**
`minimumExecutionTime`	Minimum execution time
`maximumExecutionTime`	Maximum execution time
`totalNumberOfExecutions`	Number of times operation has been executed
`averageExecutionTime`	Total execution time
`minimumMemoryRequired`	Minimum memory required
`maximumMemoryRequired`	Maximum memory required
`totalMemoryRequired`	Total memory required

The `PerformanceData` object has `get` methods that correspond to each of these properties. The average execution time and memory required can be derived from number of times and total execution time or memory required.

Exception Summary

`PerfStackMismatchException`
Thrown when `endOperation` is called with out of order arguments or different arguments than what was expected.

Using the Configuration Reporter

The ATG product suite has vast possibilities for configuration and customization. These possibilities are multiplied when you consider the different platforms, HTTP servers, and database software you might use in your site. These myriad possible combinations can make it difficult to describe your Nucleus-based web application's overall configuration in a concise way. The Configuration Reporter compiles a description of your ATG configuration, so that useful troubleshooting information is gathered in a single place.

The Configuration Reporter can generate reports in several different forms that you can use to help identify configuration problems. These reports also make it possible to e-mail configuration information to ATG support.

You can access the Configuration Reporter from the link on the Dynamo Administration UI home page, or navigate to it directly at:

```
http://hostname:port/dyn/admin/atg/dynamo/admin/en/conf-reporter.jhtml
```

The heart of the Dynamo Configuration Reporter is the service located at `/atg/dynamo/service/ConfigurationReporter`. The Configuration Reporter service works by browsing the hierarchy of components, starting at the root, gathering information, and outputting it in various formats.

Configuration Reports

The Configuration Reporter can generate the following four reports:

- HTML Component Browser Report - A report on the components in the component hierarchy in the form of HTML files. This report is more or less like printing out the entire Dynamo Administration Component Browser.
- Bean Representation Report - A list of each ATG component, with each of its properties and property values, in the form of a serialized file.
- Property Representation Report - Like the Bean Representation Report, but it includes only those components and properties whose values have been set through properties files (including properties set in the ACC).
- CONFIGPATH Report - A text file that lists the configuration path of the ATG server.

Excluding Components from the Configuration Report

By selecting a custom report, rather than a basic report, you can configure the Configuration Reporter to exclude selected components:

1. Set the `restrictedComponents` property of the `/atg/dynamo/service/ConfigurationReporter` service. This property is a comma-separated list of Nucleus component paths of components and directories that should be excluded from configuration reports.

   If a Nucleus component path included in the `restrictedComponents` property is a folder, neither it nor any of its children will be included in custom configuration reports.

2. Make a file that lists the components to include. Go to the Output Dynamo Component Hierarchy to File page at:

   ```
   http://hostname:port/dyn/admin/atg/dynamo/admin/en/
   config-reporter-output-hierarchy-titled.jhtml
   ```

3. In the **Output File** field, enter the pathname of a file to receive the list of components to include.
4. Click the **Create Dynamo Component File** button. The Configuration Reporter will generate the component list and output it to the file you specified in step 3.
5. Select **Custom Report** from the report page for the type of report you want to generate.
6. In the **Component file** field, enter the pathname of the file you created in step 4.
7. In the **Serialization output file** field, enter the pathname of a file to receive the serialized report file.
8. Click the **Create Serialization Output File** button.

The Bean Representation Report and Property Representation Report generate information in the form of serialized files. After you create a serialized report, you can output a more readable version of the information, using the XML Representation Report options:

1. Check the **Output all property values** box if you want to view the property names and values, and not just the list of components.
2. In the **Serialization output file** field, enter the name of the serialized report file you created.
3. In the **XML output file** field, enter the pathname of the file for the XML output.
4. Click the **Create XML File** button.

Running the Configuration Reporter as a Standalone Utility

You can run the Configuration Reporter as a standalone utility. This allows you to generate configuration reports even if ATG is not running. Before you run the Configuration Reporter as a standalone utility, you need to create two files:

- A file that contains a list of the components to include in the report. See Creating the Component File.
- A file that contains the configuration path. See Creating the Configuration Path File.

In addition, you should also set certain properties in `/atg/dynamo/service/ConfigurationReporter`. See Configuring the Configuration Reporter.

Creating the Component File

You can create the component file by running the report on the Output Dynamo Component Hierarchy to File page at:

```
http://hostname:port/dyn/admin/atg/dynamo/admin/en/
config-reporter-output-hierarchy-titled.jhtml
```

Add the name of the file thus created to the `componentFileName` property of `/atg/dynamo/service/ConfigurationReporter`.

As an alternative, you can create a component file by hand. The component file format is as follows:

```
<component>/Initial</component>
<component></atg/dynamo/service/Scheduler</component>
```

A component file is not expected to be well-formed XML. Anything other than what is between the component start and end tags is ignored. Anything between the tags is treated as a component name. Folders can be included between component tags; the Configuration Reporter includes all components in such a folder. Add the name of the component file to the `componentFileName` property of `/atg/dynamo/service/ConfigurationReporter`.

Creating the Configuration Path File

You can create the configuration path file by running the CONFIGPATH report on the Output Configuration Path to File page at:

```
http://hostname:port/dyn/admin/atg/dynamo/admin/en/
config-reporter-conf-path-titled.jhtml
```

Add the name of the file thus created to the `dynamoConfigurationPathFileName` property of `/atg/dynamo/service/ConfigurationReporter`.

As an alternative, you can create a configuration path file by hand. The configuration path file format is as follows:

```
<configuration_path_item>c:\ATG\ATG10.0.1\DAS\config
   </configuration_path_item>
<configuration_path_item>c:\ATG\ATG10.0.1\home\localconfig
   </configuration_path_item>
<configuration_path_item>c:\ATG\ATG10.0.1\MyModule\config
   </configuration_path_item>
```

A configuration path file is not expected to be well-formed XML. Anything other than what is between the `<configuration_path_item>` start and end tags is ignored. Anything between the tags is treated as an element of the Dynamo CONFIGPATH. Elements of the CONFIGPATH should be listed in the configuration path file in the order that they appear in the Dynamo CONFIGPATH. Add the name of the configuration path file to the dynamoConfigurationPathFileName property of /atg/dynamo/service/ConfigurationReporter.

Configuring the Configuration Reporter

As described in the previous sections, you need to set the componentFileName and dynamoConfigurationPathFileName properties of /atg/dynamo/service/ConfigurationReporter. In addition, set the serializedPropertiesFileName property to the pathname of the file you want to output.

You can set these properties using the ACC, or by adding a properties file like this at <ATG10dir>/home/localconfig/atg/service/ConfigurationReporter.properties:

```
$class=atg.service.configurationreporter.ConfigurationReader
componentFileName=
dynamoConfigurationPathFileName=
serializedPropertiesFileName=
```

Running the Configuration Reader

To run the Configuration Reporter as a standalone utility, use the following command:

```
        java atg.service.configurationreporter.ConfigurationReader
        -saveProperties config_directory
```

The *config_directory* argument is the directory that holds your ConfigurationReporter.properties file. A typical value would be localconfig.

This command generates a serialized output file. When you run this utility, the Configuration Reader reads the following input properties from properties file /atg/dynamo/service/ConfigurationReporter.properties.

dynamoConfigurationPathFileName	The name of a file that contains the Dynamo CONFIGPATH.
componentFileName	The name of the component file to read the list of Dynamo components from.
serializedPropertiesFileName	The name of the serialized file to output.

After you run the Configuration Reader utility with the -saveProperties argument, you can run it in this form to output an XML representation of the properties report:

```
        -outputRepresentationToXML SourceFile OutputFileName
            OutPutPropertyValues=true|false
```

The *SourceFile* argument is the name of the output file (serializedPropertiesFileName) and the *OutputFileName* argument is the name of the file where the Configuration Reader should output the

XML representation of the serialized output file. Use the `OutPutPropertyValues=true` flag to output the property values as well as the component names; use the `OutPutPropertyValues=false` flag to omit the property values.

Using the VMSystem Component

The ATG component located at `/VMSystem` provides a way for you to access the Java memory manager. You can monitor the status of the Virtual Machine and call methods on it. An interface to the VMSystem component is included in the Dynamo Administration UI at:

```
http://hostname:port/dyn/admin/nucleus/VMSystem/
```

From this page, you can conduct the following VM Operations:

- Perform garbage collection
- Run finalizations
- Show memory information
- List system properties
- List thread groups
- List threads
- Stop the VM

Using a Sampler

When testing your site, it is useful to automatically sample performance to understand throughput as a function of load. ATG includes a Sampler component at `/atg/dynamo/service/Sampler`. The Sampler is also discussed in the *ATG Programming Guide*.

Starting the Sampler

You can start the Sampler component by opening it in the ACC and clicking the **Start** button.

You can also start the Sampler component from the Dynamo Administration UI by requesting this URL:

```
http://hostname:port/dyn/admin/nucleus/atg/dynamo/service/Sampler
```

The first time you request this page, ATG instantiates the Sampler component, which begins recording statistics.

You can configure ATG to start the Sampler whenever ATG starts by adding the Sampler to the `initialServices` property of the `/atg/dynamo/service/Initial` component:

```
initialServices+=Sampler
```

Sampler Information

The Sampler outputs information to the file `<ATG10dir>/home/logs/samples.log`. For each system variable that it samples, it records the following information in the log file:

- the current value
- the difference between the current value and the value recorded the last minute
- the rate of change of the value

You can adjust values recorded by the Sampler, but the default set is comprehensive in monitoring ATG request handling performance. The Sampler's output includes the following:

Value	**Description**
`handledRequestCount`	Total number of requests handled by this ATG server
`averageRequestHandlingTime`	Average time spent handling requests since the sampler was started

Sampler Output

If you collect enough real data of your site under varying loads, your Sampler output gives you the answers to the following important questions:

- What is the peak throughput of your site in pages per minute for each ATG server?
- Does the peak throughput of your site go down as load increases beyond a certain threshold?
- How many sessions can each server handle while maintaining a comfortable latency (such as, latency < 1 second)?

Using the Recording Servlet

The Recording Servlet is a servlet that you place in your request handling pipeline that records the amount of time spent handling each URL on your site. It performs two distinct functions:

- Records script files used in conjunction with URLHammer. You can record scripts of actual user activity on your site, then use URLHammer to execute a script repeatedly, simulating actual system load. For information on URLHammer, see Using URLHammer in the Performance Diagnostics chapter.
- Records performance information for a single user, including the minimum, maximum, and average time spent handling each URL on your site during the recording interval.

Inserting the Recording Servlet

The Recording Servlet must be enabled before you can use it. You can enable it in one of three ways:

- Open the Recording Servlet in the ACC Component Editor at `/atg/dynamo/servlet/pipeline/RecordingServlet` and set the `recording` property to `true`.
- Request the following URL in your administration interface:

  ```
  http://hostname:port/dyn/admin/nucleus/atg/dynamo/servlet/pipeline/Record
  ingServlet
  ```

  Set the `recording` property to `true`.
- Add the Recording Servlet to the `initialServices` property of the `/atg/dynamo/servlet/Initial` component, so that the Recording Servlet is added to the servlet pipeline automatically each time your server is started:

  ```
  initialServices+=pipeline/RecordingServlet
  ```

Generating Script Files

To generate a script file from the Recording Servlet, use the Component Browser to modify the value of the `recording` property. Set this to `true` to start recording or `false` to stop recording.

Then, use your web browser to make a series of requests from your site, in the pattern of user behavior that you want to record. Each of your requests becomes part of the script.

The script is saved to the file specified by the Recording Servlet's `recordFile` property. By default, the script is saved to `<ATG10dir>/home/logs/record.log`. Each time you start recording, the old script file is overwritten. So be sure to copy the script before you enable recording for a second time.

Keeping Statistics

The Recording Servlet is also used to maintain per-URL performance statistics. To turn on this feature, set the `keepingStatistics` property to `true`. While this property is on, the minimum, maximum, and average times used to serve each requested page will be maintained and displayed in the component browser's page for the Recording Servlet component.

Tracing Memory

You can use the Recording Servlet to get an approximate reading on the amount of memory each request consumes. Set the Recording Servlet's `tracingMemory` property to true to turn on this feature. The Recording Servlet records memory information only for those URLs that run through the server one at a time; it is not appropriate for use on a live site.

8 Repository and Database Performance

Most ATG applications require database access, which represents another area where performance bottlenecks can occur. To effectively tune a large production database, your team should include an experienced database administrator.

This chapter includes the following sections:

Database Performance Practices

Follow these practices in designing and developing your site to avoid database performance problems:

- Use Repository caching features to optimize database access. See *SQL Repository Caching* in the *ATG Repository Guide*.
- Use queues to batch database transactions, rather than performing each transaction individually. See Batching Database Transactions.
- Avoid using database queries that might result in table scans of large tables. See Avoiding Table Scans.
- Run your database server on a separate machine from your application servers, or at least allocate a separate CPU.

Repositories and Transactions

By default, if you do not have a JTA transaction in place, each SQL Repository operation that affects the state of a repository item creates and commits a transaction around the operation. This is generally not the most efficient way to handle repository item updates. It is generally most efficient to ensure that all of the method calls in creating or updating a repository item are performed in a single transaction. ATG offers several different techniques for transaction demarcation that you can use to group repository method calls into a single transaction. You can use transaction demarcation in a Java Server Page using the `Transaction` servlet bean. You can demarcate a transaction programmatically. These are described in detail in the *Transaction Management* chapter of the *ATG Programming Guide*. You can also use ATG's Repository Form Handler and `TransactionalFormHandler` classes to improve the transactional behavior and performance of repository operations. See the *ATG Programming Guide* and *ATG Repository Guide* for more information.

Repository Item Property Loading

By default, whenever the SQL Repository calls `getItem`, it loads from the database (or the cache) not just the repository ID of the item, but all repository item properties that are stored in the primary database table for that item's item descriptor. For some applications, this may result in too much database activity. For other applications, you may want to load repository item properties that appear on other tables. You can adjust how the SQL Repository loads repository item properties by grouping properties, using the `group` attribute in property tags in the repository definition file. All properties with the same `group` attribute are loaded whenever one property of the group is loaded. For more information, see the *ATG Repository Guide*.

Database Sorting versus Locale-Sensitive Sorting

SQL Repository components include a `localeSensitiveSorting` property that controls how query results are sorted. If this property is set to `true`, query results are sorted using locale-sensitive String comparison (via `java.text.Collator`). Since most databases cannot handle sorting with multiple locales, setting this option to `true` also means that the repository will perform all sorting in memory. If `localeSensitiveSorting` is set to `false` (the default), database sorting (via ORDER BY) is used where applicable and Strings are compared using `String.compareTo()`. If database sorting is adequate for your purposes, leaving this property set to `false` will result in better performance. For more information, see the *ATG Repository Guide*.

Batching Database Transactions

If you have large volumes of data to insert or update, you should wherever possible perform those operations in batched transactions. It is more expensive to start a new transaction for every change than it is to attempt to make many changes in a single database transaction. For example, a request handler might log every single hit to a log table. Suppose that it takes 50 milliseconds to write a row in a log table.

If that is the case, then the request handler cannot serve requests any faster than 20 per second, even if the rest of the request handling mechanism is blazingly fast. But writing an entry in a log table is not a critical part of the request handling operation, and thus should not be such a limiting factor.

The solution to this problem is to introduce a queue between the request handler and the database facility. When the request handler wants to make a database entry, it places the log entry on the queue, then continues handling the rest of the request. A separate component reads sets of log entries and writes the whole set in a single database transaction. This arrangement decouples the request handlers from the loggers, thereby eliminating the bottleneck introduced by the database.

For more information about using queues, see the *Dynamo Foundation Classes* chapter of the *ATG Programming Guide*.

Avoiding Table Scans

A table scan is the reading of every row in a table and is caused by queries that don't properly use indexes. Table scans on large tables take an excessive amount of time and cause performance problems.

Make sure that, for any queries against large tables, **at least** one WHERE clause condition:

- refers to an indexed column and
- is reasonably selective

You should be concerned primarily with queries against large tables. If you have a table with a few hundred rows, table scans are not a problem and are sometimes faster than indexed access.

During initialization, systems like ATG may front-load caches to avoid unnecessary database operations later. You may see queries with large results during this time, but that is okay. Within reason, lengthy database operations at startup are acceptable. However, if you see frequent, large, or slow queries issuing from ATG during the course of normal operation, then you have a design problem that must be addressed to achieve acceptable performance.

For example, suppose your database has a large table that holds products such as this:

```
CREATE table product
   (    sku            char(6)        not null,
        type           char(1)        not null,
        name           varchar(50)    not null,
        description    varchar(200)   null       )
```

and has these indexes:

```
CREATE unique index i1 on product(sku)
CREATE index i2 on product(name)
CREATE index i3 on product(type)
```

The following query is fine:

```
SELECT *
  FROM product
 WHERE sku = 'a12345'
```

That query will not cause performance problems because the WHERE clause refers to a very specific condition on a column with an index.

Here is an example of a query that is likely to cause problems:

```
SELECT *
  FROM product
 WHERE description LIKE '%shoes%'
```

This query causes a table scan, since the indexes can't help the database to optimize the query. Queries like this on a large table will result in an unacceptable performance drag and therefore should not be allowed in a production system.

Here are some more queries that are likely to cause performance problems. The following query is inadvisable because, although it refers to the indexed `sku` column, it is not very selective and could return millions of rows:

```
SELECT *
  FROM product
 WHERE sku > 'abc'
```

The following query is bad because, although it is relatively selective, it will cause a table scan on most DBMSs. A LIKE query with a leading wildcard typically cannot be optimized:

```
SELECT *
  FROM product
 WHERE name LIKE '%stereo'
```

Database Caches

If you are using the SQL Repository, see how multiple requests of the same behavior affect cache usage. The first time your application references database information, the request causes a SQL database operation, but subsequent requests will use the cache. Try to optimize cache usage. Consider the best caching mode to use for each of the item descriptors in your SQL repositories. See the *ATG Repository Guide* for more information.

When you are testing the system, make sure you think about real-world usage of your data. If your system could potentially have tens of thousands or millions of rows of data, make sure you test that scenario. If you test only against small sets of data, some performance bottlenecks will be masked, because the database can cache the entire dataset into memory.

Diagnosing Database Performance Problems

Make use of performance analysis tools offered by your database and application server vendor. These tools typically enable you to measure transactions per second and memory, cache, and disk utilization. Check the CPU utilization and I/O utilization of your database server. If they are near maximum levels, this is a strong indication that the database is limiting the performance of your site.

To understand database performance, you must know your data and the operations you are performing on it. The first step is to get a copy of the DDL for all the tables in your database and get a good estimate of how many rows are in each table. Most major database systems have tools that can tell you this quickly. In a pinch, you can issue the following query for each table:

```
SELECT count(*) FROM <table-name>
```

This query might take some time for large tables, so it is best to use the vendor-supplied tools or commands. In addition to this information, you'll need a list of the indexes on each table.

Avoid Using Simulated Text Search Queries in Repositories

As a convenience feature, a SQL Repository can simulate full text searches using the SQL LIKE operator. If full text searching is not available for your database, you can substitute pattern matching queries for text search queries by setting the following property in the `GSARepository` component:

```
simulateTextSearchQueries=true
```

The SQL Repository will then convert text search queries into CONTAINS pattern match queries, which are implemented using the SQL LIKE operator.

Simulated text search queries are useful for demos and standalone development when you want to put in place the `createTextSearchQuery()` API calls without having to set up a text search engine. However, simulated text queries are extremely inefficient and are not supported for production systems. A simulated text search query using LIKE will typically cause a table scan, so you should not use simulated queries in production.

9 Tuning Site Performance on JBoss

This chapter describes configuration steps you can perform which might improve performance of your ATG software running on JBoss. Note that these are suggestions only; JBoss configuration is a complex topic, and no recommendations can be applied globally. Work with your JBoss representative to fine-tune your application's performance.

Tuning suggestions are divided into two sections:

JBoss File Modifications

JBoss Application Framework Trimming

JBoss File Modifications

This section describes changes you can make to JBoss configuration files to improve application performance.

JSP Servlet Configuration

This section concerns changes you can make to your `<JBdir>/server/`*`configdir`*`/deploy/jbossweb.deployer/conf/web.xml` file.

Add the following to the `web.xml` file under the JSP servlet (search for `<servlet-name>jsp</servlet-name>`) and make changes in that context.

```
<init-param>
     <param-name>trimSpaces</param-name>
     <param-value>false</param-value>
</init-param>
<init-param>
     <param-name>genStrAsCharArray</param-name>
     <param-value>true</param-value>
</init-param>
<init-param>
     <param-name>classDebugInfo</param-name>
     <param-value>false</param-value>
</init-param>
```

Tomcat Connector Thread Configuration

Thread pools used by Tomcat are configured on a per connector basis. The changes in this section are applied to the `<JBdir>/server/configdir/deploy/jbossweb.deployer/server.xml` file.

The default configuration is shown in this sample:

```
<!-- A HTTP/1.1 Connector on port 8080 -->
      <Connector port="8080" address="${jboss.bind.address}"
         maxThreads="250" strategy="ms" maxHttpHeaderSize="8192"
         emptySessionPath="true"
         enableLookups="false" redirectPort="8443" acceptCount="100"
         connectionTimeout="20000" disableUploadTimeout="true"/>

      <!-- Add this option to the connector to avoid problems with
          .NET clients that don't implement HTTP/1.1 correctly
         restrictedUserAgents="^.*MS Web Services Client Protocol 1.1.4322.*$"
      -->

      <!-- A AJP 1.3 Connector on port 8009 -->
      <Connector port="8009" address="${jboss.bind.address}"
         emptySessionPath="true" enableLookups="false" redirectPort="8443"
         protocol="AJP/1.3"/>
```

Thread pools can be monitored using the Tomcat monitor at `http://hostname:http_port`. The Tomcat status link is under the JBoss Management heading, for example:

```
Tomcat status (full) (XML)
```

Reducing the HTTP Connector Thread Pool

This connector is only used when you connect to Tomcat directly from your web browser. In this example, the thread pool for the HTTP connector was reduced from 250 to 20.

```
<!-- A HTTP/1.1 Connector on port 8080 -->
      <Connector port="8080" address="${jboss.bind.address}"
         maxThreads="20" strategy="lf" maxHttpHeaderSize="8192"
```

The `maxThreads` setting should reflect the expected maximum number of users that can simultaneously use the system. This number should also drive the maximum number of database connections in the datasource `*-ds.xml` file.

Full documentation for the HTTP Connector configuration can be found at `http://tomcat.apache.org`.

Increasing the AJP Connector Thread Pool

This is the primary means of contacting the server for a user (via Apache and mod_jk). In this example, the thread pool for the AJP connector is increased:

```
<!-- A AJP 1.3 Connector on port 8009 -->
      <Connector port="8009" address="${jboss.bind.address}"
          maxThreads="250" strategy="lf" minSpareThreads="50"
         emptySessionPath="true" enableLookups="false" redirectPort="8443"
         bufferSize="10240" maxHttpHeaderSize="8192" tcpNoDelay="true"
         protocol="AJP/1.3"/>
```

Full documentation for the AJP Connector and a complete dictionary of the AJP connector configuration can be found at `http://tomcat.apache.org`.

Tomcat Cluster Configuration

The `<JBdir>/server/`*`configdir`*`/deploy/jboss-web-cluster.sar/META-INF/jboss-service.xml` file contains the session replication settings. Consider the following options to improve performance:

- Use the replication strategy `REPL_ASYNC`.
- Under the UDP protocol stack ensure that the `mcast_addr` is the same on all cluster members.
- Under the UDP protocol stack ensure that the `mcast_port` is the same on all cluster members.
- When running under Windows 2003, ensure that the `loopback` attribute of the UDP protocol stack is set to `true`. For Linux this should be set to `false`. See the comment about this in the file.

JBoss Logging Configuration

JBoss uses Log4j wrapped in an MBean as a logging service. This means that an independent logging library does not need to be bundled with the application.

All logging configuration is done in the `<JBdir>/server/`*`configdir`*`/conf/jboss-log4j.xml` file. For more information on Log4j, see `http://logging.apache.org/log4j/docs/manual.html`.

You can adjust class specific logging in the `category` elements toward the end of the log4j configuration file. Each `category` can have a priority assigned to it. For example:

```
<category name="org.jboss">
   <priority value="DEBUG" />
   <appender-ref ref="FILE"/>
</category>
```

Datasource Configuration

In any `-ds.xml` files used by ATG, edit the `<min-pool-size>` and `<max-pool-size>` settings to reflect the expected maximum number of simultaneous connections.

Note: Your file may have a different name or location, depending on your configuration.

```
<min-pool-size>50</min-pool-size>
<max-pool-size>75</max-pool-size>
```

Datasource connections can be monitored using the JMX-Console at:

```
http://hostname:port/jmx-console
```

Look for ATG and ManagedConnectionFactory. The MBean monitor page shows how many connections exist and how many are being used.

Configuring run.bat/sh and run.conf

You may want to add the following JVM tuning parameters to the `JAVA_OPTS` in the `bin/run.conf` (UNIX) or `run.bat` (Windows) file:

- `-Dtomcat.util.buf.StringCache.byte.enabled=true`

  Enables the byte array to String conversion caching.

- `-Dtomcat.util.buf.StringCache.char.enabled=true`

  Enables the char array to String conversion caching.

- `-Dtomcat.util.buf.StringCache.trainThreshold=5`

  The cache is built after a training period, during which statistics about converted Strings are kept. The value of this property specifies the number of String conversions to perform before building the cache.

- `-Dtomcat.util.buf.StringCache.cacheSize=2000`

  The maximum number of String objects that will be cached, according to their usage statistics.

The effectiveness of the StringCache can be checked using the JMX-Console. Look for StringCache under Catalina in the JMX-Console page.

For the JVM command-line, the following settings can be used:

- Memory set at just over 1G for each server
- MaxPermSize adjusted to 256m

JBoss Application Framework Trimming

Removing non-required services can reduce the memory footprint as well as simplifying configuration for your application. To remove JBoss services, consider deleting the services listed below from the `deploy` (or `deploy-hasingleton`) directory.

Warning: The `jboss-service.xml` found in the *`configdir`*`/conf` directory should never be deleted or moved.

Consider whether you might be able to remove the following services:

- JBoss Mail (`mail-ra.rar`, `mail-service.xml`)
- HA-JMS (in the `deploy-hasingleton` directory of the `all` configuration)
- HA-JNDI (in `all/deploy/cluster-service.xml`, search for HAJNDI)
- UUID Key Generator (used only for CMP, `uuid-keygenerator.sar`)
- Monitoring (monitor JMX changes, in `monitoring-service.xml`)
- Scheduling (schedule tasks to execute, in `schedule-manager-service.xml` and `scheduler-service.xml`)
- EJB3 related services; see `<JBdir>/server/`*`configdir`*`/conf/jboss-service.xml`

Appendix A: Migration Issues

This chapter discusses the following topics:

Migrating from ATG 6 on WebLogic or WebSphere

If you are currently running ATG 6 on WebLogic or WebSphere and want to migrate to ATG 10.0.1, you should be aware of some significant differences between the way ATG 6 and ATG 10.0.1 applications run:

- ATG 6 uses special startup scripts and environment variables to modify the WebLogic or WebSphere system CLASSPATH so that an instance of Nucleus runs on WebLogic or WebSphere. The EAR and WAR files you build contain only standard J2EE components and configuration, while Nucleus classes and configuration remain in the ATG installation.
- ATG 9 does not modify the WebLogic or WebSphere system CLASSPATH. The applications you build are assembled into EAR files that each run their own instance of Nucleus. These EAR files include all of the class files and (optionally) configuration for the application’s Nucleus components.

When you migrate to ATG 10.0.1, you must reassemble your applications to take advantage of the new format, which is more modular, easier to maintain, and less likely to result in system resource conflicts. For information about how to reassemble your applications, see Reassembling Your Applications.

Using the JBoss Migration Tool

The JBoss Migration Tool is a Java application that is invoked through a shell script. The application automatically performs many of the steps required to transform an application that was designed to run on DAS into an application that will run on JBoss. This includes fixing JSP pages, ensuring that applications use JBoss datasources, ensuring that the correct entries are present in web.xml files, and ensuring that all EAR and WAR files are correctly listed in `MANIFEST.MF` files. The sections that follow outline the work the migration tool performs.

See the Migrating from Dynamo Application Server section of this appendix for additional changes you may want to make in your applications.

Migrating JSPs

The migration tool copies all files in the root directory you specify to the destination directory, and performs the following processing on all files ending in `.jsp` or `.jspf`, including those within `.jar` or `.zip` files:

- Substitutes all occurrences of `(DynamoHttpServletRequest)` *`[someExpression]`* with `ServletUtil.getDynamoRequest(`*`[someExpression]`*`)`, and any occurrences of `(DynamoHttpServletResponse)` *`[someExpression]`* with `(ServletUtil.getDynamoResponse)` *`[someExpression]`*.
- Looks for instances of double quotes nested within double quotes, or single quotes within single quotes, and replaces the outer quotes with single quotes or double quotes.

Migrating MANIFEST.MF

All ATG modules have a `MANIFEST.MF` file that describes properties of the module. The `ATG-EAR-Module` and `ATG-War-Module` manifest attributes specify any EAR or WAR files that should be started up in JBoss. The migration tool ensures that the `MANIFEST.MF` for all ATG modules includes references to all EAR and WAR files specified in the `J2EEContainer.properties` file within the configuration path for a given module.

Migrating web.xml Files

The migration tool searches the specified root directory, including `.jar` and `.zip` files, for all `web.xml` files. For each `web.xml` file it finds, it makes sure there is an entry for the `PageFilter` and for the `NucleusServlet`. If either is missing, the tool adds the entry to the `web.xml` and saves the modified file to the destination directory. Any `web.xml` files contained in `.jar` or `.zip` files are modified and inserted back into the copy of the jar in the destination directory.

Migrating Datasource Components

The migration tool examines all `.properties` files, `.jar` files, and `.zip` files within the ATG application's root directory, looking for configuration files that configure a `FakeXADataSource` or `MonitoredDataSource` component. It tracks entries found in `localconfig` directories separately from entries found in other locations. If a `FakeXADataSource` has any null values, then that component is ignored by the migration tool. If two datasource components have the same name and configuration path, the last one located takes priority.

After all `FakeXADataSources` and `MonitoredDataSources` have been accounted for, the migration tool creates an `atg-das-datasources-ds.xml` file in the specified JBoss server directory. If no JBoss server directory is specified, the `atg-das-datasources-ds.xml` file is created at the root level of the destination directory.

For each `FakeXADataSource` found in `localconfig`, the migration tool creates a corresponding entry in the `atg-das-datasources-ds.xml` file, giving each entry a unique JNDI name. The tool then goes through the non-`localconfig` `FakeXADataSource` components; it adds an entry for each of these to the XML file only if there is no entry in the `localconfig` map with the same component path.

Note: The migration tool is not XA-datasource aware. When it migrates existing ATG datasources to JBoss, the JBoss datasources are of type `<local-tx-datasource>`. You should edit your datasources manually to be XA datasources.

The tool then creates new configuration files for each `MonitoredDataSource`. The configuration files are placed in `home/localconfig`, using the original component path of the `MonitoredDataSource`. The new properties files differ from the original `MonitoredDataSource` properties files in the following ways:

- The `$class` property is set to `atg.nucleus.JNDIReference`.
- There is only one other property, `JNDIName`, which is set to the JNDI name of the datasource entry in the `atg-das-datasources-ds.xml` file, with a component path that matches the `MonitoredDataSource` component's original `dataSource` property value.

For example, consider the following `FakeXADataSource` component:

```
$class=atg.service.jdbc.FakeXADataSource
driver=solid.jdbc.SolidDriver
URL=jdbc:solid://localhost:1313
user=admin
password=admin
```

Along with that component is the following `MonitoredDataSource` component:

```
$class=atg.service.jdbc.MonitoredDataSource
dataSource=/atg/dynamo/service/jdbc/FakeXADataSource
```

Having found those two components, the tool would create the following entry in the `atg-das-datasources-ds.xml` file:

```
  <local-tx-datasource>
    <jndi-name>generalDS/atg/reporting/datawarehouse/FakeXADataSource
</jndi-name>
    <connection-url>jdbc:solid://localhost:1313</connection-url>
    <driver-class>solid.jdbc.SolidDriver</driver-class>
    <user-name>admin</user-name>
    <password>admin</password>
    <min-pool-size>10</min-pool-size>
    <max-pool-size>10</max-pool-size>
  </local-tx-datasource>
```

It also creates the following `MonitoredDataSource.properties` file in `home/localconfig`:

```
$class=atg.nucleus.JNDIReference
JNDIName=java\:/generalDS/atg/dynamo/service/jdbc/FakeXADataSource
```

Note: For best results, before running the migration tool, make sure that the most current and relevant datasource configs are placed in a `localconfig` directory. This will give them priority over any other datasource configs that might be found. Also, double-check the generated JBoss datasource XML file to make sure that all the datasource components are correct, and make any corrections if necessary.

Running the JBoss Migration Tool

The JBoss Migration Tool script is located in your `<ATG10dir>\DAF\JBossMigration` folder and, has the following usage syntax:

```
migrateToJBoss dynamoRootDir [-d destinationDir]
 [-j jbossServerDir] [-v]
```

It takes the following parameters:

- `dynamoRootDir`—Required. The root of the ATG application to be migrated.
- `destinationDir`—Optional. This directory will contain a copy of the source directory except for those files which have been modified for the migration. If this parameter is not specified, then a default destination directory will be created to hold the migrated files and copies of the unaltered files. The default directory will be the name of the root directory followed by the string "_migration". If a directory/file by that name already exists, then an integer will be tacked on the end of the name and incremented until a unique name is found.
- `jbossServerDir`—Optional. The location of the JBoss server deploy directory. This is needed to correctly save the generated JBoss datasource file. If you do not supply this parameter, the JBoss datasource file is saved to the root level of the `destinationDir`, and must be manually copied to the correct JBoss server directory.

Logging in JBoss Migration

As the JBoss Migration tool works, all migration actions are logged to a file in the top level of the migration destination directory. The log file contains entries for any actions that modify the original application, such as rewriting JSPs to not treat the request object as a `DynamoHttpServletRequest`, or modifying a `web.xml` to add the `PageFilter` or `NucleusServlet` entries.

Migrating from Dynamo Application Server

If you are currently running applications on DAS and want to move these applications to another application server, the migration process is straightforward. For JHTML-based applications, you should not need to make many changes to the application itself, though you will need to repackage your application.

This section discusses issues to be aware of when you migrate ATG applications from DAS to another application server.

Note: If you are migrating from DAS to JBoss, some of these steps can be performed automatically. See Using the JBoss Migration Tool.

JSP-based Applications

Because JSP-based applications rely on the application server's JSP compiler, any differences between DAS's JSP compiler and the JSP compiler on the application server you are migrating to must be taken into account. This section describes some practices to follow in your JSPs to ensure they are portable.

Using Java Expressions in Pages

- On WebLogic, the request object in a JSP is a standard `HttpServletRequest`, not a `DynamoHttpServletRequest`. To access the `DynamoHttpServletRequest` object, use `ServletUtil.getDynamoRequest(ServletRequest)`.
- On DAS, the `atg.servlet` package is imported by default in JSP pages. On other application servers, you must explicitly import the `atg.servlet` package in any page that uses classes from that package.

Using the DSP Tag Library

To learn about the DSP tag library, see the *ATG Page Developer's Guide*.

- Each JSP that uses the DSP tag library must enclose its contents in beginning and ending `dsp:page` tags, like this:

  ```
  <dsp:page>
  ... body of page ...
  </dsp:page>
  ```

- In pages that use the DSP tag library, you should avoid using standard JSP tags and instead use DSP tags wherever possible. The JSP tags do not work reliably with the DSP tag library, and the DSP tags provide additional features, such as support for the passing of object parameters between pages. In particular, use `dsp:include` rather than `jsp:include` or `jsp:forward`, and use `dsp:param` rather than `jsp:param`.
- Do not use `value="param:paramName"` to get the value of a parameter. Instead use `dsp:getvalueof` to expose a scripting variable.

JSP Syntax

- Nested pairs of quotation marks must alternate between single and double quotes. For example, the following works on DAS, but not on some application servers:

  ```
  <dsp:param name="<%="abc"%>" value="17">
  ```

  Instead, use:

  ```
  <dsp:param name='<%="abc"%>' value="17">
  ```

- Use proper syntax for concatenating text strings. For example, the following works on DAS, but not on some application servers:

  ```
  <dsp:param name="xxx"<%= "yyy" %> value="12">
  ```

  Instead, use:

  ```
  <dsp:param name='<%= "xxx" + "yyy" %>' value="12">
  ```

Servlet Pipeline

When an ATG application processes a request for a JSP that includes the DSP tag library, it invokes a servlet pipeline whose Nucleus component path is `/atg/dynamo/servlet/dafpipeline`. The DAF pipeline has fewer servlets than the DAS servlet pipeline, because it is not doing any request dispatching, mime typing, or file finding in normal operation. These operations are handled by the application server rather than by the ATG platform.

If your application adds any servlets to the DAS servlet pipeline, you will need to add these servlets to the DAF pipeline to run your application on another application server. You create a new instance of your servlet (since each pipeline servlet can be in only a single pipeline), and then insert it in the DAF servlet pipeline at the appropriate place.

There are some differences between how the DAF servlet pipeline and the DAS servlet pipeline work. For information about these differences, see *Request Handling with Servlet Pipelines* in the *ATG Programming Guide*.

Other Issues

- Use `javax.servlet.http.HttpSession`, not `atg.servlet.sessiontracking.SessionData`, as the class for your session.
- On DAS, you can use ATG's `EncodingTyper` component to specify the encoding of your JSPs. On other application servers, you must specify the encoding using the `contentType` attribute of the JSP page directive, which is the standard mechanism for defining encodings in a JSP. Note, however, that may still need to configure the `EncodingTyper` to specify the encoding of posted data in forms. See the *ATG Programming Guide* for more information.
- Do not use `request.getParameter("name")` to return parameters set using the `dsp:param` tag. Instead, use the `getDynamoRequest(request).getParameter("name")` method of the `atg.servlet.ServletUtil` class to retrieve these parameters. You can of course assume that these parameters are visible to any tags in the DSP tag library that take parameter names. Your application server's `HttpServletRequest` implementation will return only those parameters set through standard mechanisms, such as query arguments, post parameters, and parameters set using the `jsp:param` tag.
- Do not use `HttpServletRequest.getSession()` to get a session object. Instead, use the `getDynamoRequest(request).getSession()` method of the `atg.servlet.ServletUtil` class.

Migrating JHTML-based Applications

JHTML-based applications run the DAS servlet pipeline (not the DAF servlet pipeline) as a servlet in your application server. Your web application in this case must contain at least one instance of the servlet `atg.nucleus.servlet.NucleusProxyServlet`. This servlet takes an initialization parameter which is the Nucleus component path of the first servlet in the servlet pipeline. The default if no value is supplied is the string `/atg/dynamo/servlet/pipeline/DynamoHandler`, which is the Nucleus component path of the first servlet in the DAS servlet pipeline. When the `NucleusProxyServlet` receives a request, it passes it to the first servlet in the pipeline.

When you use the runAssembler command to assemble an EAR file, it includes NucleusProxyServlet in the atg_bootstrap.war web application, and includes these entries in its web.xml file:

```
<servlet>
  <servlet-name>DynamoProxyServlet</servlet-name>
  <servlet-class>atg.nucleus.servlet.NucleusProxyServlet</servlet-class>
  <load-on-startup>2</load-on-startup>
</servlet>
  ...
<servlet-mapping>
  <servlet-name>DynamoProxyServlet</servlet-name>
  <url-pattern>/dyn/*</url-pattern>
</servlet-mapping>
```

If this web application is installed in your application server with a context path of /dyn (the default), then the URLs for all JHTML pages in the application begin with:

```
http://hostname:port/dyn/dyn/
```

Note that this means that the request.getContextPath() and the request.getServletPath() methods do not return null, as they do on DAS. When configured with the servlet mapping shown above, the request.getContextPath() returns /dyn (the first one in the URL) and the request.getServletPath() returns /dyn as well (the second one).

A few servlets in the DAS servlet pipeline are disabled, because those facilities are provided by the application server (for example, the SessionServlet is disabled, because session tracking is handled by the application server). The first servlet in the pipeline creates the DynamoHttpServletRequest and Response wrappers around the HttpServletRequest and Response just as it does in DAS. All pipeline servlets you install into the servlet pipeline will work as they did in DAS.

When the request reaches the FileFinderServlet in the pipeline, this servlet translates the path to find a file relative to ATG's document root. On DAS using an ATG connection module, the connection module generally handles the translation of the paths. When you run ATG on another application server, the web server and application server cannot do the path translation, so you must configure the FileFinderServlet with all of the virtual directories used by your application. This is equivalent to how FileFinderServlet behaves on DAS when the FileFinderServlet.alwaysTranslate property is set to true.

Reassembling Your Applications

There are three main steps involved in reassembling an existing application:

1. Update the manifest files for any ATG application modules that you have created. For example, suppose your application is stored in an application module named MyApp at the top level of <ATG10dir>. You'll need to modify the <ATG10dir>/MyApp/META-INF/MANIFEST.MF file to include the manifest attributes used by the runAssembler

command. (Note that you do not need to modify the manifest files for any of the application modules that are part of the ATG installation, such as DPS and DSS. The manifest files for those modules already have all of the necessary attributes.) For more information about application modules and manifest attributes, see the *Working with Application Modules* chapter of the *ATG Programming Guide*.

2. Build an EAR file using the `runAssembler` command. For more information, see the *Developing and Assembling Nucleus-based Applications* chapter of the *ATG Programming Guide*.

3. Deploy the EAR file. Note that if you have a version of the application running, you should undeploy that version before deploying the new EAR file. See your application server documentation for information about deploying and undeploying applications.

Appendix B: Setting Up WebSphere Studio Application Developer

This appendix describes how to create ATG modules and J2EE applications in WebSphere Studio Application Developer (WSAD), and how to run those applications on WSAD's internal test server.

This appendix contains the following sections:

The procedures in this appendix assume that you have the following:

- WebSphere Studio Application Developer version installed
- ATG platform configured to run on WebSphere
- ATG Eclipse plug-in version 2.1 or above installed (see the Installing ATG Development Tools for Eclipse section in this guide)

If you are not familiar with ATG application assembly, see the *Developing and Assembling Nucleus-Based Applications* chapter in the *ATG Programming Guide*.

Creating an ATG Java Project

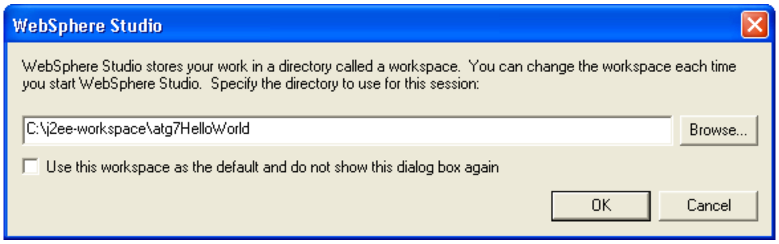
The first step in using WSAD with the ATG platform is to create an ATG module as a Java project within WSAD. The following sections explain procedures for creating a new module and importing an existing module.

You will do all of your Java development and ATG configuration in the WSAD project.

Note: Although you do not use the EAR/WAR structure of the ATG module for development purposes, it is used during assembly, so leave it intact (see Reassembling Your Application for Deployment later in this appendix).

Creating a Workspace

When you start the WSAD, specify a workspace for the J2EE projects on which you are working.



WebSphere Studio

WebSphere Studio stores your work in a directory called a workspace. You can change the workspace each time you start WebSphere Studio. Specify the directory to use for this session:

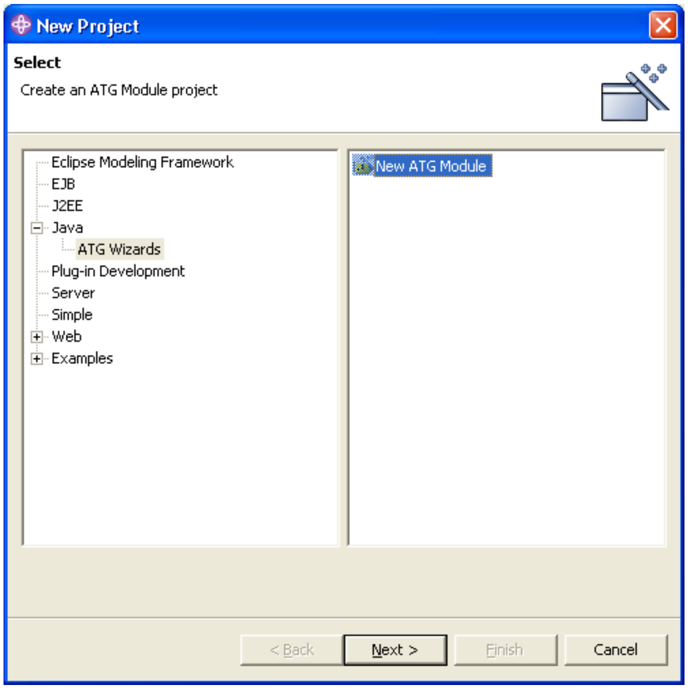
C:\j2ee-workspace\atg7HelloWorld

□ Use this workspace as the default and do not show this dialog box again

Creating a New ATG Module and WSAD Java Project

To create a new module:

1. In the WSAD, select **File** > **New** > **Project**.
2. In the folder list, select **Java** > **ATG Wizards** > **New ATG Module**.



Create an ATG Module project

3. Click **Next**.

4. Enter a name for the module.

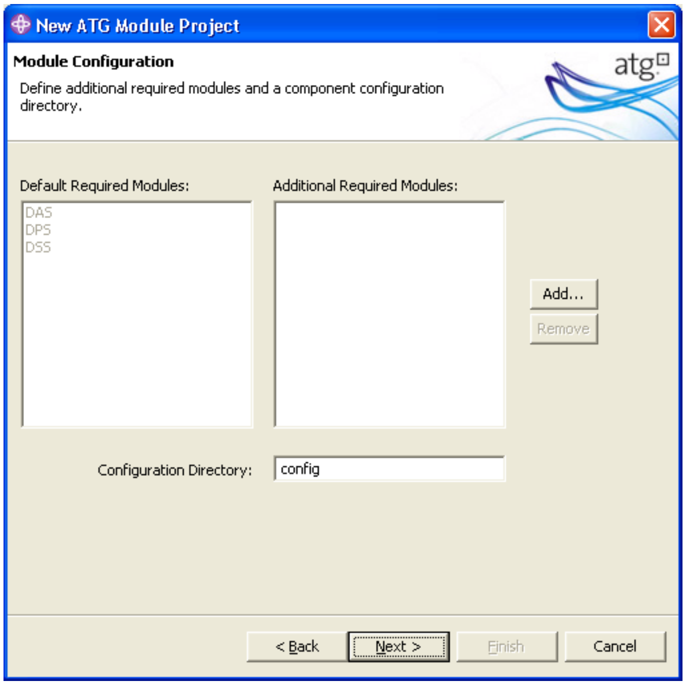
5. In the **ATG Installation: Root Directory** field, enter your `<ATG10dir>` directory. If you check **Save as Default**, this directory is used as the default root for all other ATG applications created in this workspace.

6. Click **Next**.

7. If your application requires additional ATG applications (Portal, Commerce, etc), click the **Add…** button and add them from the dialog box. For typical J2EE applications only the ATG Adaptive Scenario Engine is needed.

8. Click **Next**.

9. To add additional modules to your own module, click the **Add…** button and select modules from the dialog box. For typical J2EE applications, only DAS, DPS, and DSS are needed.



New ATG Module Project

Module Configuration

Define additional required modules and a component configuration directory.

Default Required Modules:

DAS
DPS
DSS

Additional Required Modules:

Configuration Directory: config

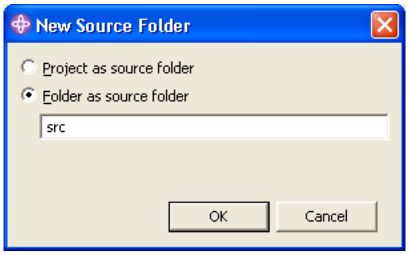
10. Leave the default Configuration Directory as **config**.

11. Click **Next**.

12. On the **Source** tab, select the displayed folder and click the **Remove** button.

13. Click **Add Folder…**.

14. On the popup, make sure that **Folder as source folder** is selected, and specify a name (such as `src`) for your Java source folder.



New Source Folder

○ Project as source folder

● Folder as source folder

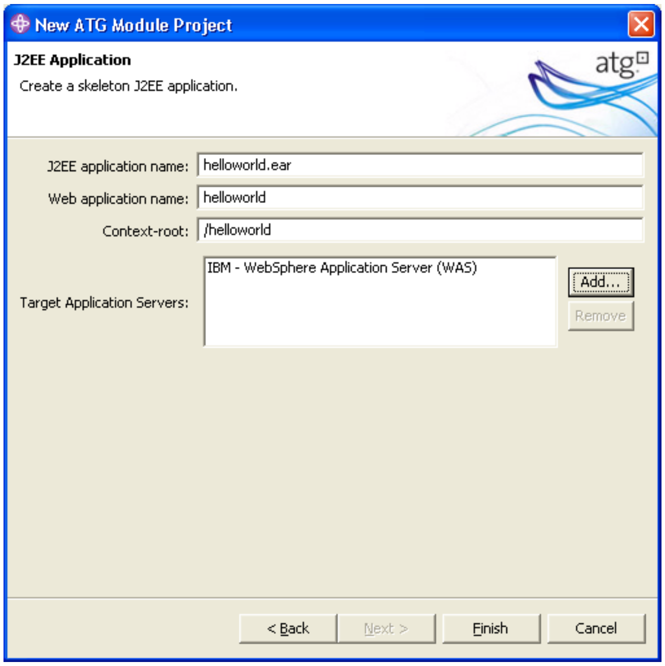
src

15. Click **OK**. (You may see an error, which is resolved in the next step.)

16. Add **\classes** to your module name in the Default Output folder field.

17. Click **Next**.

18. Enter a name for your J2EE application. It is good practice but not necessary to include the .ear extension.



J2EE Application

Create a skeleton J2EE application.

J2EE application name: helloworld.ear

Web application name: helloworld

Context-root: /helloworld

Target Application Servers: IBM - WebSphere Application Server (WAS)

19. Enter a name for your web application. Do **not** include the .war extension.

20. Specify a context root. This will be added to your `application.xml` and `web.xml`.

21. Click the **Add…** button to specify the application servers you will run on.

22. Click **Finish**.

Creating a WSAD Java Project from an Existing ATG Module

To create your project from an existing module:

1. In the WSAD, select **File** -> **New** -> **Other**.
2. In the folder list, go to **Java** > **ATG Wizards** > **Existing ATG Module**.
3. Click **Next**.
4. Enter a name for the module.
5. In the **ATG Installation: Root Directory** field, enter your `<ATG10dir>` directory. If you check **Save as Default**, this directory is used as the default root for all other ATG applications created in this workspace.
6. Click **Next**.
7. On the **Source** tab, select the displayed folder and click the **Remove** button.
8. Click **Add Folder…**. Make sure that **Folder as source folder** is checked.
9. Specify a name (such as `\src`) for your Java source folder.
10. Click **OK**.

11. Add **\classes** to your module name in the Default Output folder field.

12. Click Finish.

Generating and Importing a J2EE Application

The next step is to create an archived EAR file to import into WSAD; this creates the projects required (ear, war, ejb. etc) to take advantage of WSAD's J2EE development features.

Modifying the Manifest File

Modify the ATG-Class-Path attribute of the META-INF\MANIFEST.MF file to include a reference to classes.jar. Entries are separated by a single space. Your manifest should look similar to the following:

```
Manifest-Version: 1.0
ATG-Config-Path: config/
ATG-Required: DAS DPS DSS
ATG-J2EE: j2ee-apps/helloworld.ear
ATG-EAR-Module: j2ee-apps/helloworld.ear
ATG-Class-Path: classes lib/classes.jar <optional – dependant jars>
```

The classes.jar file is generated later in the process.

Note: It is good practice to jar up your classes when you deploy them to production rather than reference them in the classes folder.

You can add a reference to any other JAR file that your Nucleus components need access to; however, the ATG Assembler will replace forward slashes in the names with underscores, causing errors in the WSAD task list. Therefore, it is recommended that you add dependent JARs after assembling and importing your EAR (see Adding Dependent JARs).

Assembling Your J2EE Application

Use the ATG Application Assembler to generate a Development mode J2EE application file for use in WSAD (see the plug-in documentation for information on how to use the assembler).

The ATG plug-in provides a way to call the assembler:

1. In the WSAD, select **File** > **Export** > **ATG J2EE Application**.

2. From the ATG Project dropdown, select your ATG module.

3. Specify a path and name for your generated Output Ear file.

   Note: If you use the Browse option to browse the directory structure, be sure to specify the EAR name **after** you choose the appropriate folder.

1. Enter a display name (optional).

2. Select the **Pack Ear** and **Admin Console** options. Do not specify a Server.
3. Click **Finish**.

Importing the EAR file into WSAD

Import the assembled ear file into WSAD to create the needed WSAD J2EE projects.

1. In the WSAD, select **File** > **Import** > **EAR file**.
2. Browse to your previously assembled EAR file.
3. Specify a Project Name, which will be the name of your Enterprise Application project (such as MyApp_EAR).
4. Leave the Project location as your default location for this workspace, and click **Next**.
5. On the next page, leave the defaults, and click **Next**.
6. Specify names for your web project as well as the ATG generated web and EJB projects. It is good practice to use the same format you used for naming the EAR, such as MyApp_WAR or MyApp_EJB. Click **Next**.
7. For each web project, update the Java Build settings and add the ATG classes as dependant JARs; since there are references to ATG classes in the `web.xml` of each web project, the projects must know where to find the ATG classes.
    - Select each web project in the list box.
    - Click the checkbox next to the `atg_bootstrap_ejb.jar`.

    Note: You can skip this step and add a reference to these JAR files later.
8. Remove the references to any folders in the EJB classpath; you will see errors in WSAD if an EJB has a reference to anything except a JAR file. If you need access to classes in the `home/locallib` folder, jar them up and add them to your EAR's lib directory. You will get a reference to the custom module's classes later.
    - Select the `atg_bootstrap_ejb.jar`.
    - Uncheck the folder references, typically `lib/_home_slocallib` and `lib/_HelloWorld_sclasses`.
9. Click Finish.

You now have five (if you didn't choose the Admin UI option when assembling) or six WSAD projects, similar to the following list:

- `atg_bootstrap_WAR`
- `atg_bootstrap_EJB`
- `atg_admin_WAR`
- `HelloWorld`
- `helloworld_EAR`
- `helloworld_WAR`

The projects starting with atg contain the ATG framework. Helloworld_EAR is a WSAD Enterprise Application project. HelloWorld is a Java project, representing your ATG module, and is where your Java development and Nucleus component creation/configuration take place. Helloworld_WAR is a web project, and is where your JSP development takes place.

You can create Java classes in your web project if they do not need a Nucleus component (for example, a generic servlet), but creating them in your Java project simplifies things.

Setting Build References

If your web application refers to any custom classes (in a JSP or web.xml, for example), your web project needs a build reference to the classes in your Java project. To create this, modify your web project's Java Build Path settings by adding a project dependency to your Java project.

1. Right-click your web project and select **Properties**.
2. Select **Java Build Path** > **Projects** tab.
3. Check the checkbox next to your Java project.

Note: If you did not modify the manifest file to include lib/classes.jar (see Modifying the Manifest File), you can accomplish the same goal by adding a Project Reference to the atg_bootstrap_EJB project here.

Verify that the other web projects have a project reference to the atg_bootstrap_EJB project as well.

Defining a Utility JAR

The last step is to define a utility JAR for your Enterprise Application. A WSAD Utility project takes the output build directory of the chosen project, and jars it up into the file name specified when you run your application.

To create your Utility project for the lib/classes.jar file:

1. Open the EAR Deployment Descriptor of you EAR project by expanding your EAR project.
2. Click the **Module** tab.
3. Under Project Utility JARs, select **Add...**.
4. In the dialog box, select your Java project, and type in the name of the JAR file to be created. In this example it is lib/_HelloWorld_slib_sclasses.jar.

   Note: You must type the URI exactly as specified in the EJB's Manifest classpath. Notice that the forward slashes have been become underscores. You can find the EJB manifest under the EJB Project/ejbmodule/META-INF/MANIFEST.MF to verify the name of this JAR.

5. Click Finish.

Any error in your Tasks console regarding an unresolved reference to `lib/_HelloWorld_slib_sclasses.jar` should disappear.

Troubleshooting Task Console Errors

When you import a new EAR file, if you did not uncheck references to folders in the `atg_bootstrap_ejb.jar`'s dependent JAR section, you may see an error in the Tasks console referring to an entry to folders that are not resolvable (for example, `lib/_home_slocallib`, `lib/HelloWorld_sclasses`). WSAD generates this error because it only recognizes JAR files, not folders, as an EJB classpath entry. This error will not affect development.

To remove the error:

1. Open the EJB's Manifest file (`EJB Project/ejbmodule/META-INF/MANIFEST.MF`).
2. Under the Dependencies section, find the reference to these directories.
3. Uncheck this reference and save the file.

Testing Your Development Environment

Before doing any development, test your environment. The skeleton application created in the previous steps in this appendix comes with a JSP which tests Nucleus functionality.

To run your application:

1. Create a WebSphere server.
2. Start the SOLID database for the core ATG tables.
3. Generate Deployment Code for the EJB project.
4. Click **Run on Server…**.
5. Select your server.
6. Click **Finish**.

The WSAD server starts, and the console displays the output of both WebSphere and your ATG components. Once the server is started, you can open a browser and test your application by pointing to: `http://`*`hostname`*`:`*`port`*`/`*`your_context_root`* or access the Dynamo Administration UI at `http://`*`hostname`*`:`*`port`*`/dyn/admin/`.

Adding Dependent JARs

Many applications rely on third-party utility JARs for functionality. You may want to add more dependant JARs during your development process, and after you have assembled your development mode ear. In order for third-party JARs to be available to Nucleus components, they must be EAR-scoped, and should reside in the `lib` directory of your Enterprise Application. Your EJB project must also be aware of them.

Note: If you add dependent JARs to the `ATG-Class-Path` manifest attribute of your ATG module before generating the development EAR file, and the assembler distorts the names, causing errors in the WSAD task list.

To add dependant JARs to your ATG application:

1. Copy or move the necessary JAR files into your EAR project's `lib` folder. EAR project files are stored in a folder under the WSAD workspace location (for example, `C:\j2ee-workspace`).
2. In WSAD, right click your EAR project and select **Refresh**.
3. Navigate to the `\lib` directory of your EAR project. Your JARs should be visible.
4. Open the EJB's manifest file (`EJB Project/ejbmodule/META-INF/MANIFEST.MF`)
5. Under the Dependencies section, check the JAR files you want to use.
6. Save the file.

Configuring Additional ATG Servers

Create an ATG server using the `makeDynamoServer` script or Configuration Manager to use with WSAD (see Creating Additional ATG Server Instances in the Configuring Nucleus Components chapter. For example:

```
makeDynamoServer HelloWorld 9010 9011
```

This script creates a new `<ATG10>/home/servers/HelloWorld` directory with the initial subdirectories and properties files needed. Use the `/localconfig` directory to store your configuration files. This folder is added to the end of your server's configuration path, forming the final configuration layer for the components.

Configure your application to use the new server. See the *Using a Nondefault ATG Server* section in the *ATG Programming Guide*.

Reassembling Your Application for Deployment

The ATG assembler can only assemble a J2EE application from an ATG module, where the J2EE application is in a typical hierarchical structure within the module. When you import a development mode EAR file into WSAD, a new web development structure is created in your WSAD workspace, no longer using the module's J2EE application.

For that reason, you must extract the web development that you have been performing within your WSAD workspace back into your ATG module structure when assembling. This process can be done via the WSAD UI, or by using a custom ANT script.

Before you can export the web project and reassemble your application, you may need to update the `ATG-Class-Path` manifest attribute of your module if you did not do so earlier.

Reassembling Your Application Using WSAD

The first step in preparing your ATG module for assembly to a standalone EAR file is to export your web application. Before exporting an archived WAR file back into the ATG module J2EE structure, first remove the existing WAR file:

1. Navigate to your WAR file. For example: `<ATG10dir>/HelloWorld/j2ee-apps/helloworld/helloworld.war`.
2. Delete or rename the existing WAR file.
3. Select **File** > **Export** > **WAR file.**
4. Select your web project.
5. For Destination, specify the path to your ATG module's J2EE application and name your WAR file exactly as it was initially named when you first created your module (if the name of the file to be exported does not match the `<web-uri>` value in the `j2ee-apps/helloworld/META-INF/application.xml`, assembly fails). For example:

   ```
   C:\ATG\ATG10.0.1\HelloWorld\j2ee-apps\helloworld\helloworld.war
   ```

6. Select the **Overwrite** option to overwrite any existing work. Delete any old WAR files before exporting new ones.
7. Click **Finish**. You have an archived WAR file under your `j2ee-apps/ear` directory, and can assemble your application.
8. Select **File** > **Export** > **ATG J2EE Application**.
9. Select your ATG module.
10. Specify a path and name for your exported standalone EAR file. It is a good practice to have separate directories for your standalone and development EAR files.
11. Select the **Pack Ear**, **Standalone**, and **Admin Console** options. Do **not** specify a server. You must configure your application to use a non-default server instance by setting a system property, similar to setting a system property in the WSAD internal server.
12. Click **Finish**.

Reassembling Your Application Using Ant

The ATG assembler also includes two Ant tasks to simplify invoking the assembler within ant.

- `CreateUnpackedEarTask` build an unpacked/exploded EAR file
- `PackEarFileTask` archives an unpacked/exploded EAR file

See *Invoking the Application Assembler Through an Ant Task* in the *ATG Programming Guide* for information.

A generic Ant build and configuration file handles:

- Archiving your project's classes (`classes.jar`).

  Note: It does not compile your classes.
- Archiving your web project from its WSAD workspace location and placing it within your module's J2EE structure.
- Invoking the `CreateUnpackedEarTask` with the `standalone` and `adminConsole` options.
- Invoking the `PackEarFileTask`.

To copy, configure and run the ant build files:

1. Place the `build.xml` and `build.conf` in your module root (`<ATG10dir>/HelloWorld`).
2. Open the `build.conf` file.
3. Modify the properties to match your project, module, ear, and war settings. You should not need to modify `build.xml` unless you want to add functionality.
4. Open a command prompt and navigate to your module root.
5. Type `ant assemble.ear` or simply `ant` (`assemble.ear` is the default task).

Appendix C: Data Storage and Access

This appendix describes the recommended configuration for storing and accessing ATG data. It covers the following:

Database Schema Best Practices

Your DBA has ultimate control over the arrangement of ATG database schemas. However, ATG recommends as a best practice that your installation include the databases described in the following list. ATG documentation for data sources and other components uses this division as the frame of reference.

- Production Schema—Data to be accessed or affected by external users, such as product catalogs and customer profiles, and the loader tables for the data warehouse. See the Production Schema section that follows.
- Management Schema—Data required for ATG administrative applications to run, including versioned repositories and internal users. See the Management Schema section that follows.
- Agent Schema—Data to be accessed by internal users of the customer service applications, such as ATG Knowledge solutions and profile data for internal users. See the Agent Schema section that follows.
- Warehouse Schema—All of the data warehouse data. This schema should be created in a database optimized for data warehousing, and on a high-performance machine. see the *ATG Customer Intelligence Data Warehouse Guide*.

ATG documentation may also refer to a “local” schema. This schema contains the platform tables created by the `das_ddl.sql` script (see Creating the DAS Tables in this guide).

See the *ATG Multiple Application Integration Guide* for additional information on system architecture.

Production Schema

agent_profile_cmts	arf_id_generator	ARF_LOADER_PROG	ARF_LOADER_QUEUE
ARF_LQ_ENTRIES	ARF_QUEUE_ENTRY	arf_secure_id_gen	b2c_bike_owned
b2c_bike_sku	b2c_clothing_sku	b2c_compat_frame	b2c_dimensions
b2c_frame_product	b2c_item_bought	b2c_manufacturer	b2c_mnfr_keywrd
b2c_part_sku	b2c_product	b2c_sku	b2c_style
b2c_user	b2c_user_keyword	bc_campaign_track	bc_email_act_mon
bc_email_list_info	bc_email_optin	bc_email_optout	bc_imp_def_map
bc_imp_err	bc_imp_match	bc_imp_overwrite	bc_imp_prop_map
bc_imp_req	bc_import_info	bc_import_rules	bc_mailing
bc_unposted_status	bc_user_imp_aux	bc_user_imp_info	bcr_email_click
bcr_email_open	bcr_opt_out	bfr_bike_sku	bfr_clothing_sku
bfr_compat_frame	bfr_dimensions	bfr_frame_product	bfr_manufacturer
bfr_mnfr_keywrd	bfr_part_sku	bfr_product	bfr_sku
bfr_style	bjp_bike_sku	bjp_clothing_sku	bjp_compat_frame
bjp_dimensions	bjp_frame_product	bjp_manufacturer	bjp_mnfr_keywrd
bjp_part_sku	bjp_product	bjp_sku	bjp_style
caf_reg_asset	caf_reg_folder	caf_reg_pathasset	caf_reg_repasset
caf_reg_rootfolder	caf_registry	cc_campaign_track	cc_folder
cc_media	cc_media_bin	cc_media_ext	cc_media_txt
cc_usr_marker	ccr_audit_trail	ccr_campaign_entered	ccr_email_click
ccr_email_open	ccr_inbound_email	ccr_opt_out	ccr_outbound_email
csr_cc_exch_method	csr_exch	csr_exch_cmts	csr_exch_item
csr_exch_item_disp	csr_exch_items	csr_exch_method	csr_exch_methods
csr_exch_reasons	csr_exch_repl_item	csr_exch_repl_itms	csr_order_cmts

csr_return_fee	csr_sc_exch_method	csrt_ci_event	csrt_claim_item
csrt_grant_appease	csrt_oma_event	csrt_order_comment	csrt_order_event
csrt_orders	csrt_pg_event	csrt_price_overrde	csrt_recv_rtrn_itm
csrt_return_order	csrt_schd_event	csrt_sg_event	csrt_split_cc
csrt_split_sg	csrt_update_org	das_account	das_acct_prevpwd
das_cluster_name	das_dd_markers	das_dep_fail_info	das_depl_depldat
das_depl_item_ref	das_depl_options	das_depl_progress	das_depl_repmaps
das_deploy_data	das_deploy_mark	das_deployment	das_file_mark
das_group_assoc	das_gsa_subscriber	das_id_generator	das_ns_acls
das_nucl_sec	das_rep_mark	das_sds	das_secure_id_gen
das_thread_batch	dbcpp_sched_clone	dbcpp_sched_order	dcs_cart_event
dcs_cat_ancestors	dcs_cat_aux_media	dcs_cat_chldcat	dcs_cat_chldprd
dcs_cat_groups	dcs_cat_keywrds	dcs_cat_media	dcs_cat_refcfg
dcs_cat_rltdcat	dcs_catalog_refcfg	dcs_category	dcs_category_acl
dcs_catinfo_refcfg	dcs_child_fol_pl	dcs_close_qualif	dcs_complex_price
dcs_conf_options	dcs_config_opt	dcs_config_prop	dcs_discount_promo
dcs_folder	dcs_foreign_cat	dcs_gen_fol_pl	dcs_giftinst
dcs_giftitem	dcs_giftlist	dcs_giftlist_item	dcs_inventory
dcs_media	dcs_media_bin	dcs_media_ext	dcs_media_txt
dcs_ord_merge_evt	dcs_order_markers	dcs_plfol_chld	dcs_prd_ancestors
dcs_prd_aux_media	dcs_prd_chldsku	dcs_prd_groups	dcs_prd_keywrds
dcs_prd_media	dcs_prd_rltdprd	dcs_prd_skuattr	dcs_prd_upslprd
dcs_price	dcs_price_level	dcs_price_levels	dcs_price_list
dcs_prm_cls_qlf	dcs_product	dcs_product_acl	dcs_prom_used_evt
dcs_promo_grntd	dcs_promo_media	dcs_promo_rvkd	dcs_promo_upsell

dcs_promotion	dcs_refcfg_custom	dcs_refcfg_genels	dcs_refine_config
dcs_sku	dcs_sku_attr	dcs_sku_aux_media	dcs_sku_bndllnk
dcs_sku_conf	dcs_sku_link	dcs_sku_media	dcs_sku_replace
dcs_storecred_clm	dcs_submt_ord_evt	dcs_upsell_action	dcs_upsell_prods
dcs_user	dcs_user_abandoned	dcs_user_giftlist	dcs_user_otherlist
dcs_user_wishlist	dcs_usr_actvpromo	dcs_usr_promostat	dcs_usr_usedpromo
dcspp_amount_info	dcspp_amtinfo_adj	dcspp_auth_status	dcspp_bill_addr
dcspp_cc_status	dcspp_claimable	dcspp_commerce_item_markers	dcspp_config_item
dcspp_coupon	dcspp_cred_status	dcspp_credit_card	dcspp_debit_status
dcspp_det_price	dcspp_det_range	dcspp_ele_ship_grp	dcspp_gc_status
dcspp_gift_cert	dcspp_gift_inst	dcspp_giftcert	dcspp_hand_inst
dcspp_hrd_ship_grp	dcspp_item	dcspp_item_ci	dcspp_item_price
dcspp_itmprice_det	dcspp_manual_adj	dcspp_ntaxshipitem	dcspp_ord_abandon
dcspp_order	dcspp_order_adj	dcspp_order_inst	dcspp_order_item
dcspp_order_pg	dcspp_order_price	dcspp_order_rel	dcspp_order_sg
dcspp_pay_group	dcspp_pay_inst	dcspp_pay_status	dcspp_payitem_rel
dcspp_payorder_rel	dcspp_payship_rel	dcspp_price_adjust	dcspp_rel_orders
dcspp_rel_range	dcspp_relationship	dcspp_sc_status	dcspp_schd_errmsg
dcspp_sched_error	dcspp_scherr_aux	dcspp_sg_hand_inst	dcspp_ship_addr
dcspp_ship_group	dcspp_ship_inst	dcspp_ship_price	dcspp_shipitem_rel
dcspp_shipitem_sub	dcspp_shipitem_tax	dcspp_store_cred	dcspp_subsku_item

dcspp_tax_price	dcspp_taxshipitem	dfr_cat_ancestors	dfr_cat_aux_media
dfr_cat_chldcat	dfr_cat_chldprd	dfr_cat_groups	dfr_cat_keywrds
dfr_cat_media	dfr_cat_rltdcat	dfr_category	dfr_folder
dfr_media	dfr_media_bin	dfr_media_ext	dfr_media_txt
dfr_prd_ancestors	dfr_prd_aux_media	dfr_prd_chldsku	dfr_prd_groups
dfr_prd_keywrds	dfr_prd_media	dfr_prd_rltdprd	dfr_prd_skuattr
dfr_product	dfr_sku	dfr_sku_attr	dfr_sku_aux_media
dfr_sku_bndllnk	dfr_sku_link	dfr_sku_media	dfr_sku_replace
djp_cat_ancestors	djp_cat_aux_media	djp_cat_chldcat	djp_cat_chldprd
djp_cat_groups	djp_cat_keywrds	djp_cat_media	djp_cat_rltdcat
djp_category	djp_folder	djp_media	djp_media_bin
djp_media_ext	djp_media_txt	djp_prd_ancestors	djp_prd_aux_media
djp_prd_chldsku	djp_prd_groups	djp_prd_keywrds	djp_prd_media
djp_prd_rltdprd	djp_prd_skuattr	djp_product	djp_sku
djp_sku_attr	djp_sku_aux_media	djp_sku_bndllnk	djp_sku_link
djp_sku_media	djp_sku_replace	dlo_logical_org	dms_client
dms_limbo	dms_limbo_body	dms_limbo_delay	dms_limbo_msg
dms_limbo_props	dms_limbo_ptypes	dms_limbo_replyto	dms_msg
dms_msg_properties	dms_queue	dms_queue_entry	dms_queue_recv
dms_topic	dms_topic_entry	dms_topic_sub	dps_child_folder
dps_con_req	dps_con_req_sum	dps_contact_info	dps_credit_card
dps_email_address	dps_event_type	dps_folder	dps_log_id
dps_mail_batch	dps_mail_server	dps_mail_trackdata	dps_mailing
dps_markers	dps_org_ancestors	dps_org_chldorg	dps_org_role
dps_organization	dps_other_addr	dps_pgrp_con_sum	dps_pgrp_req_sum
dps_relativerole	dps_reqname_sum	dps_request	dps_role

dps_role_rel_org	dps_rolefold_chld	dps_scenario_value	dps_seg_list
dps_seg_list_folder	dps_seg_list_name	dps_session_sum	dps_user
dps_user_address	dps_user_event	dps_user_event_sum	dps_user_mailing
dps_user_org	dps_user_org_anc	dps_user_prevpwd	dps_user_roles
dps_user_scenario	dps_user_slot	dps_usr_creditcard	dps_usr_markers
drpt_conv_order	drpt_session_ord	drpt_stage_reached	dss_audit_trail
dss_coll_scenario	dss_coll_trans	dss_das_event	dss_das_form
dss_del_seg_name	dss_deletion	dss_dps_admin_prop	dss_dps_admin_reg
dss_dps_admin_up	dss_dps_click	dss_dps_event	dss_dps_inbound
dss_dps_page_visit	dss_dps_property	dss_dps_referrer	dss_dps_update
dss_dps_view_item	dss_ind_scenario	dss_ind_trans	dss_mig_info_seg
dss_mig_seg_name	dss_migration	dss_profile_slot	dss_scen_mig_info
dss_scenario_bools	dss_scenario_dates	dss_scenario_dbls	dss_scenario_info
dss_scenario_longs	dss_scenario_strs	dss_server_id	dss_slot_items
dss_slot_priority	dss_template_info	dss_user_bpmarkers	dss_xref
if_integ_data	media_base	media_bin	media_ext
media_folder	media_txt	rout_dep_hist	rout_engine
rout_env	rout_host	rout_host_inf	rout_idx_log_parts
rout_index	rout_log_part	rout_lp_cmd_count	rout_lp_smry_cmds
rout_lp_summary	rout_part	rout_phys_part_m	rout_swpchk
src_global_macro	src_roottopics_seq	src_topic	src_topic_label
src_topic_macro	src_topic_pat_seq	src_topic_pattern	src_topic_set
src_topicchild_seq	src_topicmacro_seq	srch_cfg_aprop	srch_cfg_base
srch_cfg_cfg	srch_cfg_dimnode	srch_cfg_drule	srch_cfg_dsyn

srch_cfg_dtinfo	srch_cfg_erule	srch_cfg_fol	srch_cfg_fol_chldcfgs
srch_cfg_fol_chldfol	srch_cfg_prule	srch_cfg_rank	srch_cfg_rprop
srch_cfg_rpset	srch_cfg_rrule	srch_cfg_rule	srch_cfg_synlnk
srch_cfg_synset	srch_cfg_term	srch_cfg_vrpset	srch_config
srch_config_repo	srch_refcfg_elems	srch_refel_exclude	srch_refel_order
srch_refel_range	srch_refel_select	srch_refine_config	srch_refine_elems
srch_refine_sort	srch_refine_subels	srch_sort_options	srch_update_queue
srch_update_vqueue	ssvc_logging	ssvc_prof_props	ssvc_rate_ans
ssvc_rate_event	ssvc_session_end	ssvc_ticket	ssvc_update_prof
ssvc_view_ans	svc_cell_cfg	svc_cell_def	svc_config_objct
svc_content_cfg	svc_content_def	svc_default_val	svc_fav_query
svc_fav_query_org	svc_fld_defn	svc_flddefn_bool	svc_flddefn_extaud
svc_flddefn_intaud	svc_flddefn_intmod	svc_flddefn_lval	svc_flddefn_seg
svc_fldtype_data	svc_fldval_extaud	svc_fldval_intaud	svc_fldval_intmod
svc_framewrk_cfg	svc_framewrk_def	svc_frmwk_skin	svc_frmwk_tab
svc_frmwrk_objct	svc_fw_tab_cfg	svc_fwobj_cnt	svc_fwobj_opt
svc_fwobj_tmp	svc_global_macro	svc_ksession	svc_list_value
svc_lval_intaud	svc_media_base	svc_media_bin	svc_media_ext
svc_media_folder	svc_media_txt	svc_mktg_items	svc_mktg_segments
svc_offer	svc_offer_data	svc_offer_media	svc_opt_seg
svc_org_value	svc_orgval_intaud	svc_panel_cfg	svc_panel_def
svc_ppnl_cmb	svc_pred_text	svc_ps_panels	svc_ps_pnl_cfg
svc_pstack_cfg	svc_pstack_def	svc_qoaa	svc_query
svc_query_pred	svc_rec_answer	svc_recent_tkts	svc_recommend_read

svc_renderer	svc_scls_fld_defn	svc_scls_fld_defns	svc_scls_intaud
svc_search_text	svc_seg_intaud	svc_segd_opt	svc_segdopt_info
svc_segdopt_val	svc_segment	svc_sess_view_ans	svc_session_link
svc_session_query	svc_session_reject	svc_site	svc_site_opt
svc_siteopt_info	svc_siteopt_val	svc_skin_cfg	svc_skin_def
svc_slot	svc_slt_rndrr	svc_soln	svc_soln_class
svc_soln_fld	svc_soln_int_aud	svc_soln_int_mod	svc_soln_redirect
svc_soln_segment	svc_soln_sstatus	svc_soln_status	svc_soln_topic
svc_solnfld_lnk	svc_solnfld_val	svc_solnfldval_lnk	svc_solnorg_seg
svc_solnrelevance	svc_spell_dics	svc_spell_words	svc_sstatus_tdefn
svc_tab_cells	svc_tab_cfg	svc_tab_def	svc_tab_pnl_cfg
svc_tab_pnl_def	svc_tab_psinit	svc_tab_psorder	svc_tab_pstacks
svc_template_cfg	svc_template_def	svc_tf_fldval	svc_tf_param
svc_tf_param_val	svc_tfldval_lnk	svc_tfparam_lval	svc_tfplval_lnk
svc_topic	svc_topic_label	svc_topic_macro	svc_topic_pat_seq
svc_topic_pattern	svc_topicchild_seq	svc_topiclabel	svc_topicmacro_seq
svc_topicusecount	svc_user_favorites	svc_user_opt	svc_useropt_info
svc_useropt_val	svc_usr_loginbrand	svc_usr_mktg_sgmts	svc_usr_srch
svc_viewed_answer	svc_window_attrb	svct_callnote_act	svct_prob_cat
svct_research_act	tkt_act_escal	tkt_act_map	tkt_act_message
tkt_act_ownagnt	tkt_act_owngrp	tkt_act_pcreate	tkt_act_pswchange
tkt_act_statc	tkt_act_worknote	tkt_activity	tkt_ads_act_data
tkt_ads_in_msgs	tkt_ads_messages	tkt_ads_mms_msgs	tkt_ads_msg_addrs
tkt_ads_msg_atts	tkt_ads_msg_hdrs	tkt_ads_msg_props	tkt_ads_msgaddlist
tkt_ads_msgattlist	tkt_ads_out_msgs	tkt_ads_pop3_msgs	tkt_ads_raw_msgs

tkt_ads_sms_msgs	tkt_ads_smtp_msgs	tkt_attachment	tkt_attch_list
tkt_cust_details	tkt_dist_srv_stat	tkt_esc_own_group	tkt_esc_tkt_q
tkt_ext_ref	tkt_extref_list	tkt_owning_group	tkt_q_stat_set
tkt_q_stats	tkt_queue	tkt_rea_context	tkt_rea_ctx_list
tkt_reason	tkt_related	tkt_sub_status	tkt_ticket
tkt_upd_props	tkt_update_prof		

Management Schema

alt_chan_usr_rel	alt_channel	alt_gear	alt_gear_def
alt_gear_def_rel	alt_gear_rel	alt_group	alt_user
alt_user_alert_rel	alt_user_pref	alt_userpref_rel	avm_asset_lock
avm_devline	avm_workspace	bc_action	bc_action_attr
bc_action_param	bc_array	bc_array_const	bc_attribute
bc_campaign_data	bc_campaign_note	bc_cond_attr	bc_cond_clause
bc_cond_clause_flt	bc_condition	bc_constant	bc_email_optin
bc_email_optout	bc_event	bc_event_attr	bc_event_filter
bc_event_prop	bc_event_prop_name	bc_expression	bc_filter
bc_flt_cond_clause	bc_flt_operand	bc_folder	bc_imp_def_map
bc_imp_err	bc_imp_match	bc_imp_overwrite	bc_imp_prop_map
bc_imp_req	bc_import_info	bc_import_rules	bc_item
bc_jndi_prop	bc_jndi_prop_name	bc_mailing	bc_nucl_prop_name
bc_nucleus_prop	bc_parameter	bc_participant_flt	bc_profile_grp_flt
bc_servlet	bc_st_cond_clause	bc_subj_prop_name	bc_subject_prop
bc_user_imp_aux	bc_user_imp_info	bc_var_prop_name	bc_variable
bc_variable_prop	bcr_control	bcr_email_click	bcr_email_open

bcr_opt_out	bcr_time_dim	bcr_time_dim_meta	caf_reg_asset
caf_reg_folder	caf_reg_pathasset	caf_reg_repasset	caf_reg_rootfolder
caf_registry	cc_action	cc_action_part_flt	cc_campaign_data
cc_email_comm	cc_email_comm_evt	cc_email_comm_lp	cc_event
cc_event_wait	cc_exit_campaign	cc_fill_slot	cc_fill_slot_cont
cc_fill_slot_event	cc_folder	cc_gen_act_event	cc_gen_action
cc_land_page_event	cc_landing_page	cc_list_import	cc_media
cc_media_bin	cc_media_ext	cc_media_txt	cc_stage
cc_stage_action	cc_stage_preempt	cc_time_wait	cc_usr_marker
ccr_audit_fact	ccr_audit_trail	ccr_campaign_entered	ccr_email_click
ccr_email_fact	ccr_email_open	ccr_inbound_email	ccr_lndpg_fact
ccr_opt_out	ccr_optout_fact	ccr_outbound_email	comm_gear_add
comm_gear_rem	das_account	das_acct_prevpwd	das_cluster_name
das_dd_markers	das_dep_fail_info	das_depl_depldat	das_depl_item_ref
das_depl_options	das_depl_progress	das_depl_repmaps	das_deploy_data
das_deploy_mark	das_deployment	das_file_mark	das_group_assoc
das_gsa_subscriber	das_id_generator	das_ns_acls	das_nucl_sec
das_rep_mark	das_sds	das_secure_id_gen	das_thread_batch
dbcpp_sched_clone	dbcpp_sched_order	dcs_cart_event	dcs_cat_ancestors
dcs_cat_aux_media	dcs_cat_chldcat	dcs_cat_chldprd	dcs_cat_groups
dcs_cat_keywrds	dcs_cat_media	dcs_cat_rltdcat	dcs_category
dcs_category_acl	dcs_child_fol_pl	dcs_close_qualif	dcs_complex_price
dcs_conf_options	dcs_config_opt	dcs_config_prop	dcs_discount_promo
dcs_folder	dcs_foreign_cat	dcs_gen_fol_pl	dcs_giftinst
dcs_giftitem	dcs_giftlist	dcs_giftlist_item	dcs_inventory
dcs_media	dcs_media_bin	dcs_media_ext	dcs_media_txt

dcs_ord_merge_evt	dcs_order_markers	dcs_plfol_chld	dcs_prd_ancestors
dcs_prd_aux_media	dcs_prd_chldsku	dcs_prd_groups	dcs_prd_keywrds
dcs_prd_media	dcs_prd_rltdprd	dcs_prd_skuattr	dcs_prd_upslprd
dcs_price	dcs_price_level	dcs_price_levels	dcs_price_list
dcs_prm_cls_qlf	dcs_product	dcs_product_acl	dcs_prom_used_evt
dcs_promo_grntd	dcs_promo_media	dcs_promo_rvkd	dcs_promo_upsell
dcs_promotion	dcs_sku	dcs_sku_attr	dcs_sku_aux_media
dcs_sku_bndllnk	dcs_sku_conf	dcs_sku_link	dcs_sku_media
dcs_sku_replace	dcs_storecred_clm	dcs_submt_ord_evt	dcs_upsell_action
dcs_upsell_prods	dcs_user	dcs_user_abandoned	dcs_user_giftlist
dcs_user_otherlist	dcs_user_wishlist	dcs_usr_actvpromo	dcs_usr_promostat
dcs_usr_usedpromo	dcspp_amount_info	dcspp_amtinfo_adj	dcspp_auth_status
dcspp_bill_addr	dcspp_cc_status	dcspp_claimable	dcspp_commerce_item_markers
dcspp_config_item	dcspp_coupon	dcspp_cred_status	dcspp_credit_card
dcspp_debit_status	dcspp_det_price	dcspp_det_range	dcspp_ele_ship_grp
dcspp_gc_status	dcspp_gift_cert	dcspp_gift_inst	dcspp_giftcert
dcspp_hand_inst	dcspp_hrd_ship_grp	dcspp_item	dcspp_item_ci
dcspp_item_price	dcspp_itmprice_det	dcspp_manual_adj	dcspp_ntaxshipitem
dcspp_ord_abandon	dcspp_order	dcspp_order_adj	dcspp_order_inst
dcspp_order_item	dcspp_order_pg	dcspp_order_price	dcspp_order_rel
dcspp_order_sg	dcspp_pay_group	dcspp_pay_inst	dcspp_pay_status
dcspp_payitem_rel	dcspp_payorder_rel	dcspp_payship_rel	dcspp_price_adjust
dcspp_rel_orders	dcspp_rel_range	dcspp_relationship	dcspp_sc_status
dcspp_schd_errmsg	dcspp_sched_error	dcspp_scherr_aux	dcspp_sg_hand_inst

dcspp_ship_addr	dcspp_ship_group	dcspp_ship_inst	dcspp_ship_price
dcspp_shipitem_rel	dcspp_shipitem_sub	dcspp_shipitem_tax	dcspp_store_cred
dcspp_subsku_item	dcspp_tax_price	dcspp_taxshipitem	dms_client
dms_limbo	dms_limbo_body	dms_limbo_delay	dms_limbo_msg
dms_limbo_props	dms_limbo_ptypes	dms_limbo_replyto	dms_msg
dms_msg_properties	dms_queue	dms_queue_entry	dms_queue_recv
dms_topic	dms_topic_entry	dms_topic_sub	dpi_access_right
dpi_child_folder	dpi_contact_info	dpi_email_address	dpi_folder
dpi_mail_batch	dpi_mail_server	dpi_mail_trackdata	dpi_mailing
dpi_org_ancestors	dpi_org_chldorg	dpi_org_role	dpi_organization
dpi_other_addr	dpi_relativerole	dpi_role	dpi_role_rel_org
dpi_role_right	dpi_rolefold_chld	dpi_scenario_value	dpi_template_role
dpi_user	dpi_user_address	dpi_user_mailing	dpi_user_org
dpi_user_org_anc	dpi_user_prevpwd	dpi_user_roles	dpi_user_scenario
dpi_user_sec_orgs	dpi_user_slot	dps_child_folder	dps_con_req
dps_con_req_sum	dps_contact_info	dps_credit_card	dps_email_address
dps_event_type	dps_folder	dps_log_id	dps_mail_batch
dps_mail_server	dps_mail_trackdata	dps_mailing	dps_markers
dps_org_ancestors	dps_org_chldorg	dps_org_role	dps_organization
dps_other_addr	dps_pgrp_con_sum	dps_pgrp_req_sum	dps_relativerole
dps_reqname_sum	dps_request	dps_role	dps_role_rel_org
dps_rolefold_chld	dps_scenario_value	dps_seg_list	dps_seg_list_folder
dps_seg_list_name	dps_session_sum	dps_user	dps_user_address
dps_user_event	dps_user_event_sum	dps_user_mailing	dps_user_org
dps_user_org_anc	dps_user_prevpwd	dps_user_roles	dps_user_scenario
dps_user_slot	dps_usr_creditcard	dps_usr_markers	drpt_conv_order

drpt_session_ord	drpt_stage_reached	dsi_coll_scenario	dsi_coll_trans
dsi_del_seg_name	dsi_deletion	dsi_ind_scenario	dsi_ind_trans
dsi_mig_info_seg	dsi_mig_seg_name	dsi_migration	dsi_profile_slot
dsi_scen_mig_info	dsi_scenario_bools	dsi_scenario_dates	dsi_scenario_dbls
dsi_scenario_info	dsi_scenario_longs	dsi_scenario_strs	dsi_server_id
dsi_slot_items	dsi_slot_priority	dsi_template_info	dsi_xref
dss_audit_trail	dss_coll_scenario	dss_coll_trans	dss_das_event
dss_das_form	dss_del_seg_name	dss_deletion	dss_dps_admin_prop
dss_dps_admin_reg	dss_dps_admin_up	dss_dps_click	dss_dps_event
dss_dps_inbound	dss_dps_page_visit	dss_dps_property	dss_dps_referrer
dss_dps_update	dss_dps_view_item	dss_ind_scenario	dss_ind_trans
dss_mig_info_seg	dss_mig_seg_name	dss_migration	dss_profile_slot
dss_scen_mig_info	dss_scenario_bools	dss_scenario_dates	dss_scenario_dbls
dss_scenario_info	dss_scenario_longs	dss_scenario_strs	dss_server_id
dss_slot_items	dss_slot_priority	dss_template_info	dss_user_bpmarkers
dss_xref	epub_agent	epub_agent_trnprt	epub_binary_file
epub_coll_workflow	epub_dep_err_parm	epub_dep_log	epub_deploy_proj
epub_deployment	epub_dest_map	epub_exclud_asset	epub_file_asset
epub_file_folder	epub_his_act_parm	epub_history	epub_includ_asset
epub_ind_workflow	epub_int_prj_hist	epub_int_user	epub_pr_history
epub_pr_tg_ap_ts	epub_pr_tg_dp_id	epub_pr_tg_dp_ts	epub_pr_tg_status

epub_princ_asset	epub_prj_targt_ws	epub_prj_tg_snsht	epub_proc_history
epub_proc_prv_prj	epub_proc_taskinfo	epub_process	epub_process_data
epub_project	epub_target	epub_taskinfo	epub_text_file
epub_tl_targets	epub_topology	epub_tr_agents	epub_tr_dest
epub_wf_coll_trans	epub_wf_del_segs	epub_wf_deletion	epub_wf_ind_trans
epub_wf_mg_inf_seg	epub_wf_mig_info	epub_wf_mig_segs	epub_wf_migration
epub_wf_server_id	epub_wf_templ_info	epub_workflow_bls	epub_workflow_dats
epub_workflow_dbls	epub_workflow_info	epub_workflow_lngs	epub_workflow_ris
epub_workflow_strs	epub_workflow_vfs	if_integ_data	media_base
media_bin	media_ext	media_folder	media_txt
mem_membership_req	paf_base_comm_role	paf_base_gear_role	paf_cf_child_item
paf_cf_gfldrs	paf_child_folder	paf_col_palette	paf_comm_gears
paf_comm_gfldrs	paf_comm_ldescs	paf_comm_lnames	paf_comm_roles
paf_comm_template	paf_community	paf_community_acl	paf_cpal_ln_descs
paf_cpal_ln_names	paf_ct_alt_gear	paf_ct_alt_gr_rel	paf_ct_child_fldr
paf_ct_folder	paf_ct_gear	paf_ct_gears	paf_ct_gr_acl
paf_ct_gr_fldrs	paf_ct_gr_iparams	paf_ct_gr_ln_descs	paf_ct_gr_ln_names
paf_ct_gr_roles	paf_ct_page	paf_ct_pagefolder	paf_ct_pg_regions
paf_ct_region	paf_ct_region_grs	paf_ct_roles	paf_device_output
paf_device_outputs	paf_display_modes	paf_fldr_ln_descs	paf_fldr_ln_names
paf_folder	paf_folder_acl	paf_gd_cprops	paf_gd_iparams
paf_gd_l10n_descs	paf_gd_l10n_names	paf_gd_uparams	paf_gdf_child_item
paf_gear	paf_gear_acl	paf_gear_def	paf_gear_iparams

paf_gear_ln_descs	paf_gear_ln_names	paf_gear_modes	paf_gear_param
paf_gear_prmvals	paf_gear_roles	paf_layout	paf_layout_regdefs
paf_page	paf_page_acl	paf_page_ln_descs	paf_page_ln_names
paf_page_regions	paf_page_template	paf_page_visit	paf_pf_child_item
paf_ptpl_ln_descs	paf_ptpl_ln_names	paf_region	paf_region_def
paf_region_gears	paf_styl_ln_descs	paf_styl_ln_names	paf_style
paf_template	paf_title_template	page_gear_add	page_gear_rem
vmap_attrval	vmap_attrval_rel	vmap_cattrval_rel	vmap_fh
vmap_im	vmap_im2ivm_rel	vmap_iv	vmap_iv2ivad_rel
vmap_ivattrdef	vmap_ivm	vmap_ivm2pvm_rel	vmap_mode
vmap_pv	vmap_pv2pvad_rel	vmap_pvattrdef	vmap_pvm

Agent Schema

agent_audit	agent_call	agent_org_props	agent_prof_props
agent_session_end	agent_update_org	agent_update_prof	alt_chan_usr_rel
alt_channel	alt_gear	alt_gear_def	alt_gear_def_rel
alt_gear_rel	alt_group	alt_user	alt_user_alert_rel
alt_user_pref	alt_userpref_rel	avm_asset_lock	avm_devline
avm_workspace	caf_reg_asset	caf_reg_folder	caf_reg_pathasset
caf_reg_repasset	caf_reg_rootfolder	caf_registry	comm_gear_add
comm_gear_rem	csr_ci_event	csr_claim_item	csr_grant_appease
csr_oma_event	csr_order_comment	csr_order_event	csr_pg_event
csr_price_override	csr_recv_rtrn_item	csr_return_order	csr_schd_event

csr_sg_event	csr_split_cc	csr_split_sg	csr_upd_props
csr_view_card	das_account	das_acct_prevpwd	das_cluster_name
das_dd_markers	das_dep_fail_info	das_depl_depldat	das_depl_item_ref
das_depl_options	das_depl_progress	das_depl_repmaps	das_deploy_data
das_deploy_mark	das_deployment	das_file_mark	das_group_assoc
das_gsa_subscriber	das_id_generator	das_ns_acls	das_nucl_sec
das_rep_mark	das_sds	das_secure_id_gen	das_thread_batch
dms_client	dms_limbo	dms_limbo_body	dms_limbo_delay
dms_limbo_msg	dms_limbo_props	dms_limbo_ptypes	dms_limbo_replyto
dms_msg	dms_msg_properties	dms_queue	dms_queue_entry
dms_queue_recv	dms_topic	dms_topic_entry	dms_topic_sub
dpi_access_right	dpi_child_folder	dpi_contact_info	dpi_email_address
dpi_folder	dpi_mail_batch	dpi_mail_server	dpi_mail_trackdata
dpi_mailing	dpi_org_ancestors	dpi_org_chldorg	dpi_org_role
dpi_organization	dpi_other_addr	dpi_relativerole	dpi_role
dpi_role_rel_org	dpi_role_right	dpi_rolefold_chld	dpi_scenario_value
dpi_template_role	dpi_user	dpi_user_address	dpi_user_mailing
dpi_user_org	dpi_user_org_anc	dpi_user_prevpwd	dpi_user_roles
dpi_user_scenario	dpi_user_sec_orgs	dpi_user_slot	dps_child_folder
dps_con_req	dps_con_req_sum	dps_contact_info	dps_email_address
dps_event_type	dps_folder	dps_log_id	dps_mail_batch
dps_mail_server	dps_mail_trackdata	dps_mailing	dps_markers
dps_org_ancestors	dps_org_chldorg	dps_org_role	dps_organization
dps_other_addr	dps_pgrp_con_sum	dps_pgrp_req_sum	dps_relativerole
dps_reqname_sum	dps_request	dps_role	dps_role_rel_org

dps_rolefold_chld	dps_scenario_value	dps_seg_list	dps_seg_list_folder
dps_seg_list_name	dps_session_sum	dps_user	dps_user_address
dps_user_event	dps_user_event_sum	dps_user_mailing	dps_user_org
dps_user_org_anc	dps_user_prevpwd	dps_user_roles	dps_user_scenario
dps_user_slot	dps_usr_markers	drpt_stage_reached	dsi_coll_scenario
dsi_coll_trans	dsi_del_seg_name	dsi_deletion	dsi_ind_scenario
dsi_ind_trans	dsi_mig_info_seg	dsi_mig_seg_name	dsi_migration
dsi_profile_slot	dsi_scen_mig_info	dsi_scenario_bools	dsi_scenario_dates
dsi_scenario_dbls	dsi_scenario_info	dsi_scenario_longs	dsi_scenario_strs
dsi_server_id	dsi_slot_items	dsi_slot_priority	dsi_template_info
dsi_xref	dss_audit_trail	dss_coll_scenario	dss_coll_trans
dss_das_event	dss_das_form	dss_del_seg_name	dss_deletion
dss_dps_admin_prop	dss_dps_admin_reg	dss_dps_admin_up	dss_dps_click
dss_dps_event	dss_dps_inbound	dss_dps_page_visit	dss_dps_property
dss_dps_referrer	dss_dps_update	dss_dps_view_item	dss_ind_scenario
dss_ind_trans	dss_mig_info_seg	dss_mig_seg_name	dss_migration
dss_profile_slot	dss_scen_mig_info	dss_scenario_bools	dss_scenario_dates
dss_scenario_dbls	dss_scenario_info	dss_scenario_longs	dss_scenario_strs
dss_server_id	dss_slot_items	dss_slot_priority	dss_template_info
dss_user_bpmarkers	dss_xref	epub_agent	epub_agent_trnprt
epub_binary_file	epub_coll_workflow	epub_dep_err_parm	epub_dep_log
epub_deploy_proj	epub_deployment	epub_dest_map	epub_exclud_asset
epub_file_asset	epub_file_folder	epub_his_act_parm	epub_history
epub_includ_asset	epub_ind_workflow	epub_int_prj_hist	epub_int_user

epub_pr_history	epub_pr_tg_ap_ts	epub_pr_tg_dp_id	epub_pr_tg_dp_ts
epub_pr_tg_status	epub_princ_asset	epub_prj_targt_ws	epub_prj_tg_snsht
epub_proc_history	epub_proc_prv_prj	epub_proc_taskinfo	epub_process
epub_process_data	epub_project	epub_target	epub_taskinfo
epub_text_file	epub_tl_targets	epub_topology	epub_tr_agents
epub_tr_dest	epub_wf_coll_trans	epub_wf_del_segs	epub_wf_deletion
epub_wf_ind_trans	epub_wf_mg_inf_seg	epub_wf_mig_info	epub_wf_mig_segs
epub_wf_migration	epub_wf_server_id	epub_wf_templ_info	epub_workflow_bls
epub_workflow_dats	epub_workflow_dbls	epub_workflow_info	epub_workflow_lngs
epub_workflow_ris	epub_workflow_strs	epub_workflow_vfs	if_integ_data
media_base	media_bin	media_ext	media_folder
media_txt	mem_membership_req	paf_base_comm_role	paf_base_gear_role
paf_cf_child_item	paf_cf_gfldrs	paf_child_folder	paf_col_palette
paf_comm_gears	paf_comm_gfldrs	paf_comm_ldescs	paf_comm_lnames
paf_comm_roles	paf_comm_template	paf_community	paf_community_acl
paf_cpal_ln_descs	paf_cpal_ln_names	paf_ct_alt_gear	paf_ct_alt_gr_rel
paf_ct_child_fldr	paf_ct_folder	paf_ct_gear	paf_ct_gears
paf_ct_gr_acl	paf_ct_gr_fldrs	paf_ct_gr_iparams	paf_ct_gr_ln_descs
paf_ct_gr_ln_names	paf_ct_gr_roles	paf_ct_page	paf_ct_pagefolder
paf_ct_pg_regions	paf_ct_region	paf_ct_region_grs	paf_ct_roles
paf_device_output	paf_device_outputs	paf_display_modes	paf_fldr_ln_descs
paf_fldr_ln_names	paf_folder	paf_folder_acl	paf_gd_cprops
paf_gd_iparams	paf_gd_l10n_descs	paf_gd_l10n_names	paf_gd_uparams

paf_gdf_child_item	paf_gear	paf_gear_acl	paf_gear_def
paf_gear_iparams	paf_gear_ln_descs	paf_gear_ln_names	paf_gear_modes
paf_gear_param	paf_gear_prmvals	paf_gear_roles	paf_layout
paf_layout_regdefs	paf_page	paf_page_acl	paf_page_ln_descs
paf_page_ln_names	paf_page_regions	paf_page_template	paf_page_visit
paf_pf_child_item	paf_ptpl_ln_descs	paf_ptpl_ln_names	paf_region
paf_region_def	paf_region_gears	paf_styl_ln_descs	paf_styl_ln_names
paf_style	paf_template	paf_title_template	page_gear_add
page_gear_rem	srch_cfg_aprop	srch_cfg_base	srch_cfg_cfg
srch_cfg_dimnode	srch_cfg_drule	srch_cfg_dsyn	srch_cfg_dtinfo
srch_cfg_erule	srch_cfg_fol	srch_cfg_fol_chldcfgs	srch_cfg_fol_chldfol
srch_cfg_prule	srch_cfg_rank	srch_cfg_rprop	srch_cfg_rpset
srch_cfg_rrule	srch_cfg_rule	srch_cfg_synlnk	srch_cfg_synset
srch_cfg_term	srch_cfg_vrpset	srch_config	srch_config_repo
srch_refcfg_elems	srch_refel_exclude	srch_refel_order	srch_refel_range
srch_refel_select	srch_refine_config	srch_refine_elems	srch_refine_sort
srch_refine_subels	srch_sort_options	srch_update_queue	srch_update_vqueue
svc_cell_cfg	svc_cell_def	svc_config_objct	svc_content_cfg
svc_content_def	svc_default_val	svc_fld_defn	svc_flddefn_bool
svc_flddefn_extaud	svc_flddefn_intaud	svc_flddefn_intmod	svc_flddefn_lval
svc_flddefn_seg	svc_fldtype_data	svc_fldval_extaud	svc_fldval_intaud
svc_fldval_intmod	svc_framewrk_cfg	svc_framewrk_def	svc_frmwk_skin
svc_frmwk_tab	svc_frmwrk_objct	svc_fw_tab_cfg	svc_fwobj_cnt
svc_fwobj_opt	svc_fwobj_tmp	svc_list_value	svc_lval_intaud
svc_media_base	svc_media_bin	svc_media_ext	svc_media_folder

svc_media_txt	svc_mktg_items	svc_mktg_segments	svc_offer
svc_offer_data	svc_offer_media	svc_opt_seg	svc_org_value
svc_orgval_intaud	svc_panel_cfg	svc_panel_def	svc_ppnl_cmb
svc_process_data	svc_ps_panels	svc_ps_pnl_cfg	svc_pstack_cfg
svc_pstack_def	svc_qoaa	svc_renderer	svc_scls_fld_defn
svc_scls_fld_defns	svc_scls_intaud	svc_seg_intaud	svc_segd_opt
svc_segdopt_info	svc_segdopt_val	svc_segment	svc_site
svc_site_opt	svc_siteopt_info	svc_siteopt_val	svc_skin_cfg
svc_skin_def	svc_slot	svc_slt_rndrr	svc_soln
svc_soln_class	svc_soln_fld	svc_soln_int_aud	svc_soln_int_mod
svc_soln_segment	svc_soln_status	svc_soln_taskinfo	svc_soln_topic
svc_solnfld_lnk	svc_solnfld_val	svc_solnfldval_lnk	svc_status_right
svc_tab_cells	svc_tab_cfg	svc_tab_def	svc_tab_pnl_cfg
svc_tab_pnl_def	svc_tab_psinit	svc_tab_psorder	svc_tab_pstacks
svc_template_cfg	svc_template_def	svc_tf_fldval	svc_tf_param
svc_tf_param_val	svc_tfldval_lnk	svc_tfparam_lval	svc_tfplval_lnk
svc_user_opt	svc_useropt_info	svc_useropt_val	svcm_attachments
svcm_batch_step	svcm_classprops	svcm_fav_solutions	svcm_group
svcm_named_acl	svcm_og_val_map	svcm_property	svcm_soln_class
svcm_step	svcm_stmt_security	svcm_user	svcm_user_favs
svcm_userfavs_m	svcr_search_env	svcr_sol_event	svcr_ticket_event
tkt_org_tktqs	tkt_push_agent	vmap_attrval	vmap_attrval_rel
vmap_cattrval_rel	vmap_fh	vmap_im	vmap_im2ivm_rel
vmap_iv	vmap_iv2ivad_rel	vmap_ivattrdef	vmap_ivm
vmap_ivm2pvm_rel	vmap_mode	vmap_pv	vmap_pv2pvad_rel
vmap_pvattrdef	vmap_pvm		

Data Sources

The following table lists the datasource components available for use by ATG applications. The datasources you configure will depend on which applications you are using.

Data Source Component Name	Module Defined In	Configured In
/atg/dynamo/service/jdbc/JTDataSource This datasource is always configured to point to the core schema for the server instance on which it is running. For instance, on a Content Administration server, it points to the management schema; on a production server, it points to the production schema.	DAS	config
/atg/dynamo/service/jdbc/JTDataSource_production This datasource points to a production schema, but runs on a non-production server instance, such as asset management or agent.	DAS	config
/atg/dynamo/service/jdbc/JTDataSource_staging This datasource points to a staging schema, but runs on a non-staging server instance, such as asset management, production, or agent.	DafEar.base	configlayers/stagingandprod
/atg/reporting/datawarehouse/loaders/JTDataSource	ARF.base	config
/atg/reporting/datawarehouse/JTDataSource	ARF.DW.base	config
/atg/commerce/jdbc/ProductCatalogSwitchingDataSource Used for switching. See Configuring a SwitchingDataSource in this guide.	DCS	config
/atg/commerce/jdbc/ProductCatalogDataSourceA Used for switching. See Configuring a SwitchingDataSource in this guide.	DCS	config

`/atg/commerce/jdbc/ProductCatalogDataSourceB` Used for switching. See Configuring a SwitchingDataSource in this guide.	`DCS`	`config`
`/atg/search/service/SearchJTDataSource`	`DAF.Search.Base`	`config`
`/atg/dynamo/service/jdbc/JTDataSource_agent`	`DAS`	`config`
`/atg/dynamo/service/jdbc/JTDataSource_management`	`DAS`	`config`
`/atg/dynamo/service/jdbc/eServerJTDataSource`	`Service.migration`	`config`
`/atg/dynamo/service/jdbc/SelfServiceReportingJTDataSource`	`Service.SelfServiceDataWarehouse`	`config`
`/atg/campaign/communication/reporting/JTDataSource`	`ACO.communication.DW`	`config`

Repositories

- /atg/search/routing/repository/SearchConfigurationRepository (Routing in the diagram)
- /atg/search/repository/IncrementalItemQueueRepository.properties (Indexing in the diagram)

The following table lists the repositories used by ATG applications, which datasource they use by default, and server-dependent conditions for use:

Repository Component Name	**Datasource Component** **Configuration Varies Depending on Server**
`/atg/commerce/catalog/ProductCatalog`	`/atg/dynamo/service/JTDataSource_production`
`/atg/commerce/atg/ClaimableRepository`	`/atg/dynamo/service/JTDataSource_production`
`/atg/commerce/gifts/GiftLists`	`/atg/dynamo/service/JTDataSource_production`
`/atg/commerce/inventory/InventoryRepository`	`/atg/dynamo/service/JTDataSource_production`

/atg/commerce/jdbc/ProductCatalogDataSourceA	/atg/dynamo/service/JTDataSource_production
/atg/commerce/order/OrderRepository	/atg/dynamo/service/JTDataSource_production
/atg/commerce/pricing/priceLists/PriceLists	/atg/dynamo/service/JTDataSource_production
/atg/scenario/ScenarioClusterManager	/atg/dynamo/service/JTDataSource_production
/atg/userprofiling/ProfileAdapterRepository	/atg/dynamo/service/JTDataSource_production
/atg/userprofiling/PersonalizationRepository	/atg/dynamo/service/JTDataSource_production
/atg/search/service/SearchJTDataSource	Generic Reference to /atg/dynamo/service/JTDataSource_production
/atg/search/repository/RefinementRepository	/atg/dynamo/service/JTDataSource_production
/atg/search/SearchTestingRepository	/atg/dynamo/service/JTDataSource_management
/atg/search/routing/repository/SearchConfigurationRepository	/atg/search/service/SearchJTDataSource
/atg/search/repository/IncrementalItemQueueRepository	/atg/dynamo/service/JTDataSource_production
/atg/searchadmin/TopicRepository	/atg/search/service/SearchJTDataSource
/atg/content/media/MediaRepository	/atg/dynamo/service/JTDataSource
/atg/dynamo/messaging/SqlJmsProvider	/atg/dynamo/service/JTDataSource
/atg/dynamo/service/ClusterName	/atg/dynamo/service/JTDataSource
/atg/dynamo/service/IdGenerator	/atg/dynamo/service/JTDataSource
/atg/dynamo/service/ObfuscatedIdGenerator	/atg/dynamo/service/JTDataSource
/atg/dynamo/service/jdbc/SDSRepository	/atg/dynamo/service/JTDataSource_production
/atg/dynamo/service/jdbc/SQLRepository	/atg/dynamo/service/JTDataSource
/atg/integrations/repository/IntegrationsRepository	/atg/dynamo/service/JTDataSource

/atg/webservice/security/NucleusSecurityRepository	/atg/dynamo/service/JTDataSource
/atg/compaign/communication/OutreachRepository	/atg/dynamo/service/JTDataSource_production
/atg/epub/PublishingRepository	/atg/dynamo/service/JTDataSource_management
/atg/epub/process/ProcessDataRepository	/atg/dynamo/service/JTDataSource_management
/atg/epub/process/VersionManagerRepository	/atg/dynamo/service/JTDataSource_management
/atg/epub/process/PortalRepository	/atg/dynamo/service/JTDataSource_production
/atg/userprofiling/InternalProfileRepository	/atg/dynamo/service/JTDataSource_management
/atg/dynamo/service/jdbc/BCRJTDataSource	/atg/dynamo/service/JTDataSource_management
/atg/reporting/datawarehouse/LogicalOrganizationReportRepository	/atg/reporting/datawarehouse/JTDataSource
/atg/reporting/datawarehouse/RmClsRoutingReportRepository	/atg/reporting/datawarehouse/JTDataSource
/atg/reporting/datawarehouse/RMReportRepository	/atg/reporting/datawarehouse/JTDataSource
/atg/userprofiling/InternalProfileRepository	/atg/dynamo/service/JTDataSource_agent
/atg/commerce/custsvc/CsrRepository	/atg/dynamo/service/JTDataSource_production
/atg/svc/ServiceRepository	/atg/dynamo/service/JTDataSource_production
/atg/svc/ui/framework/ServiceFrameworkRepository	/atg/dynamo/service/JTDataSource_production
/atg/svc/option/UserOptionRepository	/atg/dynamo/service/JTDataSource_production
/atg/svc/option/OptionRepository	/atg/dynamo/service/JTDataSource_production
/atg/svc/userprofiling/ServiceSegmentRepository	/atg/dynamo/service/JTDataSource_production

/atg/svc/shared/ServiceSharedRepository	/atg/dynamo/service/JTDataSource_production
/atg/svc/service/ServiceJTDataSource	/atg/dynamo/service/JTDataSource_production
/atg/svc/logging/SelfServiceLoggingRepository	/atg/dynamo/service/JTDataSource_production

Appendix D: Adjusting the FileCache Size

ATG's servlet pipeline includes servlets that are used for JHTML pages, and which use a `FileCache` component to store files that ATG has read from disk, so that subsequent accesses for those files can be delivered directly from memory instead of being read from disk. Using the `FileCache` component improves performance by reducing disk accesses. For maximum performance, you want the `FileCache` to be large enough to hold all the files that ATG serves frequently. Set the `totalSize` property of this component at:

```
/atg/dynamo/servlet/pipeline/FileCache
```

to an appropriate value, measured in bytes, such as the following:

```
# size in bytes (2 million bytes)
totalSize=2000000
```

One approach in sizing the `FileCache` is to batch compile the entire document root and set the file cache to the resulting size. Make sure, however, that you account for the size of your `FileCache` when you set the size of your JVM. You can preload the `FileCache` by creating a script that accesses every page on your site and running the script on startup.

You can view statistics on how the file cache is used, as well as the contents of the `FileCache` in the Dynamo Administration page at *`hostname:port`*`/dyn/admin/nucleus/atg/dynamo/servlet/pipeline/FileCache`.

Index