

daytrader - a more complex application

{scrollbar}

DayTrader is benchmark application built around the paradigm of an online stock trading system. Originally developed by IBM as the Trade Performance Benchmark Sample, DayTrader was donated to the Apache Geronimo community in 2005. This application allows users to login, view their portfolio, lookup stock quotes, and buy or sell stock shares. With the aid of a Web-based load driver such as Mercury LoadRunner, Rational Performance Tester, or Apache JMeter, the real-world workload provided by DayTrader can be used to measure and compare the performance of Java Platform, Enterprise Edition (Java EE) application servers offered by a variety of vendors.

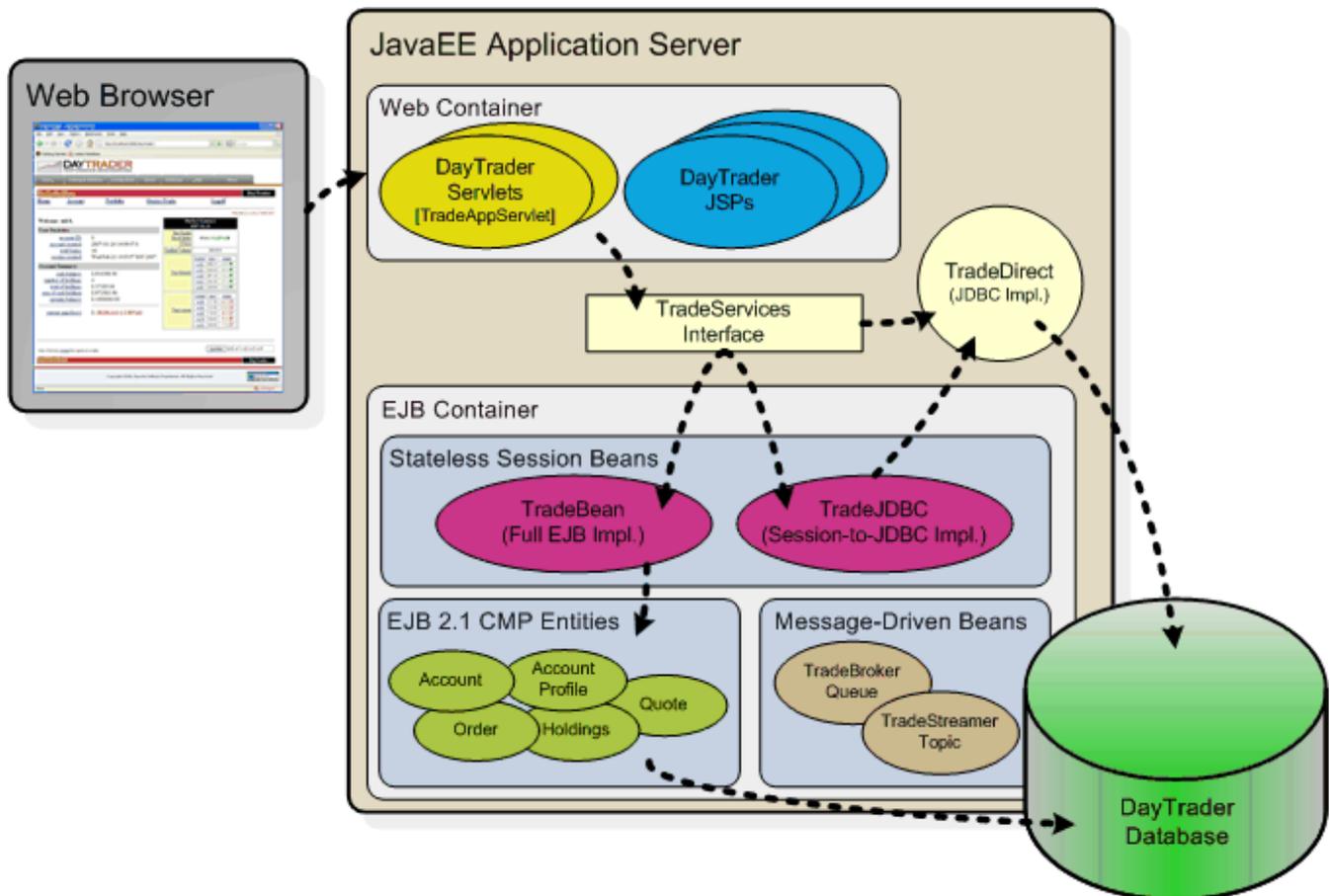
In addition to the full workload, the application also contains a set of primitives used for functional and performance testing of various Java EE components and common design patterns.

This document is organized in the following sections:

3

Application Architecture

DayTrader is built on a core set of Java EE technologies that includes Java Servlets and JavaServer Pages (JSPs) for the presentation layer and Java database connectivity (JDBC), Java Message Service (JMS), Enterprise JavaBeans (EJBs) and Message-Driven Beans (MDBs) for the back-end business logic and persistence layer. The following diagram provides a high-level overview of the full workload application architecture.



Presentation Layer

The presentation layer consists of several Java Servlets and JSPs that loosely adhere to a Model-View-Controller (MVC) design pattern. *TradeAppServlet* is the primary controller servlet responsible for receiving incoming client requests, triggering the desired business logic, and forwarding responses to the appropriate JSP page. Additional servlets and JSPs are used to configure the DayTrader runtime options and manage the supporting database.

Business Logic and Persistence Layer

The business logic and persistence layer form the bulk of the DayTrader application. The *TradeServices* interface defines the core set of business operations available in the application, such as register, login, getHoldings, buy, completeOrder, logout, etc. DayTrader provides three different implementations of these services, corresponding to three commonly used JavaEE application design patterns. These implementations are discussed below. Users can switch between these implementations on the configuration page by changing the Runtime Mode.

Implementation	Details
TradeDirect (Default)	Pattern: Servlet-to-JDBC Runtime Mode: Direct The <i>TradeDirect</i> class performs CRUD (create, read, update, and delete) operations directly against the supporting database using custom JDBC code. Database connections, commits, and rollbacks are managed manually in the code. JTA user transactions are used to coordinate 2-phase commits.
TradeJDBC	Pattern: Servlet-to-SessionBean-to-JDBC Runtime Mode: Session Direct The <i>TradeJDBC</i> stateless session bean serves as a wrapper for TradeDirect. The session bean assumes control of all transaction management while TradeDirect remains responsible for handling the JDBC operations and connections. This implementation reflects the most commonly used JavaEE application design pattern.
TradeBean	Pattern: Servlet-to-SessionBean-to-EntityBean Runtime Mode: EJB The <i>TradeBean</i> stateless session bean uses Container Managed Persistence (CMP) entity beans to represent the business objects. The state of these objects is completely managed by the application servers EJB container.

Another subtle component of this layer involves the Java Messaging Service (JMS). JMS is used within DayTrader for two specific purposes, asynchronously processing buy/sell orders, and publishing quote price updates. The following table discusses these operations in further detail.

Operation	Details
Asynchronous Order Processing	When a buy or sell operation is performed, an order request is placed on the TradeBroker JMS queue using a client connection. The TradeBrokerMDB consumes messages on this queue and completes the buy or sell operation.
Quote Price Updates	As stocks are traded, the associated quote prices are updated in the database and published to a JMS topic. The TradeStreamerMDB subscribes to these updates consuming the price updates messages, but does nothing more with them. The TradeStreamer JavaEE client that is bundled with DayTrader can be started to view the quote prices updates in real time.

Business Objects and Relationships

The following diagram represents the database schema and associated business objects. Container managed relationships (CMRs) are also depicted in the diagram.

Create diagram and add here

Business Operations (as defined in TradeServices)

As previously mentioned, all of the primary business operations provided by DayTrader are defined in the TradeServices interface. These operations are discussed further in the following table.

TradeServices Operation	Details
login	
logout	
buy	
sell	
getMarketSummary	
queueOrder	
completeOrder	
cancelOrder	
orderCompleted	
getOrders	
getClosedOrders	
createQuote	
getQuote	

getAllQuotes	
updateQuotePriceVolume	
getHoldings	
getHolding	
getAccountData	
getAccountProfileData	
updateAccountProfile	
register	
resetTrade	

User Interface (UI) Operations

The DayTrader JSP/Servlet-based web client provides a basic set of operations that one would expect to find in any stock trading and portfolio management application. These high level user operations trigger specific business operations (defined above) within the business logic and persistence layers to perform the desired task. The following table summarizes the business tasks performed by each user operation/action.

Client (UI) Operation	Flow of Business Operations
Register	
Login	
View Account	
View Account Profile	
Update Account Profile	
View Portfolio	
Sell Holding	
View Quotes	
Buy Stock	
Logout	

Getting the source

Daytrader is available in the Apache's subversion repository, run the following command to checkout the source files into the **daytrader-2.0** directory.

```
svn co http://svn.apache.org/repos/asf/geronimo/daytrader/trunk/ <daytrader_home>
```

<daytrader_home> could be any directory dedicated to hold daytrader-2.0.

This process may take several minutes depending on the machine and network connectivity speed.

Building Daytrader

Once all the sources get checked out the next step is to build Daytrader. Daytrader requires Maven 2 for building the binaries.

From the <daytrader_home> directory run the following command.

```
mvn install
```

This process will take a couple of minutes. The binaries will be generated in the corresponding **target** directory for each of the modules in the **modules** directory.

Configuring Daytrader

By default Daytrader requires a database to be created using the embedded Derby database that is shipped with Geronimo. Typically, the provided deployment plan files are configured to create such database (DaytraderDatabase) on Apache Derby during deployment. However, scripts are provided within the <daytrader_home>/bin/dbscripts/derby directory to create this database manually. Note that at this point this step **optional**, you can still create the required database after deploying Daytrader and using the **(Re)-create DayTrader Database Tables and Indexes** link from the application's **Configuration Utility** page.

Independently on whether you use the command line scripts or the web based option, you will need the tables created before getting to the [#Populating sample data](#) section.

The purpose of this section is to show you how to use the provided scripts to create the required **DaytraderDatabase** so, if needed, you can adapt them to your specific configuration environment. Additional scripts for different databases are also provided.

- Start Geronimo by running the following command:
`<geronimo_home>/bin/geronimo start`
- The provided database creation script requires setting the **GERONIMO_HOME** environment variable. On the same window you start Geronimo run the following command:
`set GERONIMO_HOME=<geronimo_home>`
- Change directory to the directory containing the database creation scripts.
`cd <daytrader_home>/bin/dbscripts/derby`
- Open `createDerbyDB` script and verify/modify the Derby version to match the one being used by Geronimo (e.g. `<geronimo_home>/repository/org/apache/derby/derby/10.2.2.0`). Once you verified the versions match run the script.
`createDerbyDB`
You should see a scree similar to the one illustrated below. solid D:\daytrader-2.0\bin\dbscripts\derby>createDerbyDB.bat "Invoking IJ command line tool to create the database and tables...please wait" ij version 10.2.2.0 ij> ij> ERROR 42Y55: 'DROP TABLE' cannot be performed on 'HOLDINGEJB' because it does not exist. ij> ERROR 42Y55: 'DROP TABLE' cannot be performed on 'ACCOUNTPROFILEEJB' because it does not exist. ij> ERROR 42Y55: 'DROP TABLE' cannot be performed on 'QUOTE EJB' because it does not exist. ij> ERROR 42Y55: 'DROP TABLE' cannot be performed on 'ACCOUNTEJB' because it does not exist. ij> ERROR 42Y55: 'DROP TABLE' cannot be performed on 'ORDEREJB' because it does not exist. ij> 0 rows inserted /updated/deleted ij> 0 rows inserted/updated/deleted ij> 0 rows inserted/updated/deleted ij> 0 rows inserted/updated/deleted ij> 0 rows inserted /updated/deleted ij> 0 rows inserted/updated/deleted ij> 0 rows inserted/updated/deleted ij> 0 rows inserted /updated/deleted ij> ij> Table creation complete
- You can verify the database was created by pointing your browser to the Geronimo Administration Console and clicking on **DB Manager**.
- The last step in this configuration is to update the deployment plan. Edit the `daytrader-g-2.0-SNAPSHOT-plan.xml` deployment plan located in the `<daytrader_home>\plans` directory and replace `ge-activemq-rar/1.2-beta/rar` with `ge-activemq-rar/1.2/rar`.

You are now ready to deploy the application.

Deploying Daytrader

So far we have retrieved the source file, built, configured, created a database and updated the deployment plan. Now it is time to install the Daytrader application in Geronimo.

There are basically two ways to deploy an application in Geronimo, either using the Geronimo Administration Console or the command line based deployer tool. For this example we will be using the command line based option.

From the `<geronimo_home>/bin` directory run the following command:

```
deploy --user system --password manager deploy <daytrader_home>\modules\ear\target\daytrader-ear-2.0-SNAPSHOT.ear <daytrader_home>\plans\daytrader-g-2.0-SNAPSHOT-plan.xml
```

The first `deploy` is the script that calls the deployer tool, then we pass the user name and password. The second `deploy` is the actual command option for deploying the `daytrader-ear-2.0-SNAPSHOT.ear` EAR using the `daytrader-g-2.0-SNAPSHOT-plan.xml` deployment plan specifically. In your own application you could call this plan `geronimo-application.xml` and place it in the `META-INF` directory within you EAR file and you will not need to expressly specify the deployment plan from the command line.

You should see a deployment confirmation screen similar to the one shown below.

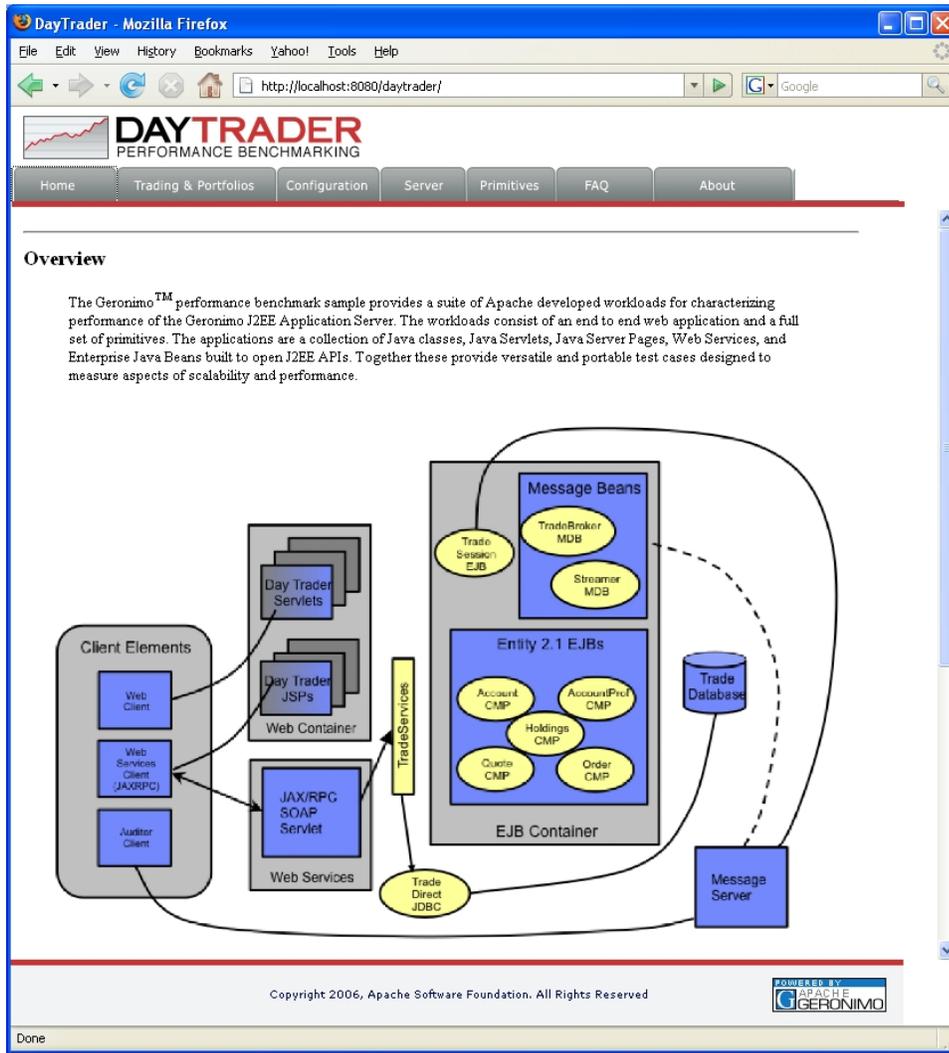
```
solid D:\geronimo-tomcat6-javaee5-2.1\bin>deploy --user system --password manager deploy \daytrader-2.0\modules\ear\target\daytrader-ear-2.0-SNAPSHOT.ear \daytrader-2.0\plans\daytrader-g-2.0-SNAPSHOT-plan.xml Using GERONIMO_BASE: D:\geronimo-tomcat6-javaee5-2.1 Using GERONIMO_HOME: D:\geronimo-tomcat6-javaee5-2.1 Using GERONIMO_TMPDIR: D:\geronimo-tomcat6-javaee5-2.1\var\temp Using JRE_HOME: C:\Java\jdk1.5.0_06\jre Deployed geronimo/daytrader/2.0-SNAPSHOT/car -> web.war @ http://localhost:8080/daytrader -> dt-ejb.jar -> geronimo/daytrader-wsapp-client/2.0-SNAPSHOT/car -> geronimo/daytrader-streamer-client/2.0-SNAPSHOT/car -> TradeDataSource -> TradeJMS
```

Daytrader is now ready for testing.

Populating sample data

With the application deployed and started (starts by default when you deploy it) the next step before using Daytrader is to populate sample data to the database we created before. The following steps illustrate how.

- Access the application pointing your browser to <http://localhost:8080/daytrader>



- Click on the **Configuration** tab.
- Click on **(Re)-populate DayTrader Database** to generate the sample data, this will open a new window showing the progress. The initial population size consists of 200 accounts and 400 stock quotes. These values can be updated via the "Configure DayTrader run-time parameters" link on the "Configuration" tab.

Running Daytrader

Daytrader can be run in number of configurations and also provides a suite of web primitives to ease testing. Each of these primitives singularly test key operations in the enterprise Java programming model. Some of these can be configured to run repeatedly based on the configuration settings that we will cover later on. The following sections describe more in detail these primitives test suite.

Web Container ping suite

The following table describe the Web container related set of primitives. Those primitives that can be set to run multiple times are **highlighted**.

Primitive	Description
PingHtml	PingHtml is the most basic operation providing access to a simple "Hello World" page of static HTML.
Explicit GC	Invoke Garbage Collection on AppServer. Reports heap statistics after the GC has completed.
PingServlet	PingServlet tests fundamental dynamic HTML creation through server side servlet processing.
PingServlet Writer	PingServletWriter extends PingServlet by using a PrintWriter for formatted output vs. the output stream used by PingServlet.
PingServlet 2Include	PingServlet2Include tests response inclusion. Servlet 1 includes the response of Servlet 2.

PingServlet2Servlet	PingServlet2Servlet tests request dispatching. Servlet 1, the controller, creates a new JavaBean object forwards the request with the JavaBean added to Servlet 2. Servlet 2 obtains access to the JavaBean through the Servlet request object and provides dynamic HTML output based on the JavaBean data.
PingJSP	PingJSP tests a direct call to JavaServer Page providing server-side dynamic HTML through JSP scripting.
PingJSPEL	PingJSPEL tests a direct call to JavaServer Page providing server-side dynamic HTML through JSP scripting and the usage of the new JSP 2.0 Expression Language.
PingServlet2JSP	PingServlet2JSP tests a commonly used design pattern, where a request is issued to servlet providing server side control processing. The servlet creates a JavaBean object with dynamically set attributes and forwards the bean to the JSP through a RequestDispatcher The JSP obtains access to the JavaBean and provides formatted display with dynamic HTML output based on the JavaBean data.
PingHTTPSession1	PingHTTPSession1 - <code>sessionID</code> tests fundamental HTTP session function by creating a unique session ID for each individual user. The ID is stored in the users session and is accessed and displayed on each user request.
PingHTTPSession2	PingHTTPSession2 <code>session create/destroy</code> further extends the previous test by invalidating the HTTP Session on every 5th user access. This results in testing HTTPSession create and destroy.
PingHTTPSession3	PingHTTPSession3 <code>large session object</code> tests the servers ability to manage and persist large HTTPSession data objects. The servlet creates a large custom java object. The class contains multiple data fields and results in 2048 bytes of data. This large session object is retrieved and stored to the session on each user request.
PingJDBCRead	PingJDBCRead tests fundamental servlet to JDBC access to a database performing a single-row read using a prepared SQL statement.
PingJDBCWrite	PingJDBCRead tests fundamental servlet to JDBC access to a database performing a single-row write using a prepared SQL statement.
PingServlet2JNDI	PingServlet2JNDI tests the fundamental J2EE operation of a servlet allocating a JNDI context and performing a JNDI lookup of a JDBC DataSource.

EJB Container ping suite

The following table describe the EJB container related set of primitives. Those primitives that can be set to run multiple times are **highlighted**.

Primitive	Description
PingServlet2SessionEJB	PingServlet2SessionEJB tests key function of a servlet call to a stateless SessionEJB. The SessionEJB performs a simple calculation and returns the result.
PingServlet2EntityEJBLocal PingServlet2EntityEJBRemote	PingServlet2EntityEJB tests key function of a servlet call to an EJB 2.0 Container Managed Entity. In this test the EJB entity represents a single row in the database table. The <code>Local version</code> uses the EJB Local interface while the <code>Remote version</code> uses the Remote EJB interface. (Note: PingServlet2EntityEJBLocal will fail in a multi-tier setup where the Trade Web and EJB apps are separated.)
PingServlet2Session2Entity	This tests the full servlet to Session EJB to Entity EJB path to retrieve a single row from the database.
PingServlet2Session2EntityCollection	This test extends the previous EJB Entity test by calling a Session EJB which uses a finder method on the Entity that returns a collection of Entity objects. Each object is displayed by the servlet
PingServlet2Session2CMROne2One	This test drives an Entity EJB to get another Entity EJB's data through an EJB 2.0 CMR One to One relationship
PingServlet2Session2CMROne2Many	This test drives an Entity EJB to get another Entity EJB's data through an EJB 2.0 CMR One to Many relationship
PingServlet2Session2JDBC	This tests the full servlet to Session EJB to JDBC path to retrieve a single row from the database.
PingServlet2Session2JDBCcollection	This test extends the previous JDBC test by calling a Session EJB to JDBC path which returns multiple rows from the database.
PingServlet2MDBQueue	PingServlet2MDBQueue drives messages to a Queue based Message Driven EJB (MDB).Each request to the servlet posts a message to the Queue. The MDB receives the message asynchronously and prints message delivery statistics on each 100th message. Note: Not intended for performance testing.
PingServlet2MDBTopic	PingServlet2MDBTopic drives messages to a Topic based Publish/Subscribe Message Driven EJB (MDB).Each request to the servlet posts a message to the Topic. The TradeStreamMDB receives the message asynchronously and prints message delivery statistics on each 100th message. Other subscribers to the Topic will also receive the messages. Note: Not intended for performance testing.
PingServlet2TwoPhase	PingServlet2TwoPhase drives a Session EJB which invokes an Entity EJB with <code>findByPrimaryKey</code> (DB Access) followed by posting a message to an MDB through a JMS Queue (Message access). These operations are wrapped in a global 2-phase transaction and commit.

Configure DayTrader run-time

So now you know what set of primitives are available and which of those can be set to run multiple times. The following table describes what parameters are available from the Daytrader **Configuration Utilities** to set the runtime parameters.

Parameter	Option	Description
Run-Time Mode	EJB Direct SessionDir ect JPA	Run Time Mode determines server implementation of the TradeServices to use in the DayTrader application Enterprise Java Beans including Session, Entity and Message beans or Direct mode which uses direct database and JMS access.
Order-Processing Mode	Synchrono us Asynchron ous_2- Phase	Order Processing Mode determines the mode for completing stock purchase and sell operations. Synchronous mode completes the order immediately. Asynchronous_2-Phase performs a 2-phase commit over the EJB Entity/DB and MDB/JMS transactions.
Access Mode	Standard WebServic es	Access Mode determines the protocol used by the DayTrader Web application to access server side services. The Standard mode uses the default Java RMI protocol. The Web Services mode uses the Axis implementation of Web Services including SOAP, WSDL and UDDI.
Scenario Workload Mix	Standard High- Volume	This setting determines the runtime workload mix of DayTrader operations when driving the benchmark through TradeScenarioServlet.
WebInterface	JSP JSP- Images	This setting determines the Web interface technology used, JSPs or JSPs with static images and GIFs.

Miscellaneous Settings

DayTrader Max Users Trade Max Quotes	By default the DayTrader database is populated with 50 users (uid:0 - uid:49) and 100 quotes (s:0 - s:99).
Primitive Iteration	By default the DayTrader primitives are execute one operation per web request. Change this value to repeat operations multiple times per web request.
Publish Quote Updates	Publish quote price changes to a JMS topic. Needed for running the #Streamer application client .
Enable long run support	Enable long run support by disabling the show all orders query performed on the Account page.
Enable operation trace Enable full trace	Enable DayTrader processing trace messages.

Running Primitives

So far we saw what primitives are available, which of those can be set to run multiple iterations and how to configure the application runtime parameters.

- Point your browser to <http://localhost:8080/daytrader>
- Click on **Configuration**.
- Click on **Configure DayTrader run-time parameters**.
- Select **EJB** from **Run-Time Mode**.
- Select **JSP-Images** from **WebInterface**.
- Set **Primitive Iteration** to 100.
- Click on **Update Config**.
- Click on **Primitives**.
- Click on **PingServlet2EntityEJBLocal**.

With these settings, every time you hit **PingServlet2EntityEJBLocal** or refresh the page that primitive will get executed 100 times. When doing performance analysis, being able to "play" with these parameters is very valuable. This helps you track down execution times of these very specific functions. When used combined with a load simulation tool, the different configurations will assist you with the fine tuning of the server based on the specific needs of your environment.

Gone trading !!!

We just saw how to run singular functions/operations tests via the available primitives. The very same settings you configured for running those primitives also affect the GUI for trading simulation.

- Point your browser to <http://localhost:8080/daytrader>
- Click on **Trading & Portfolios**.
- Accept the default user and password and click on **Login**.
- You should now be able to begin trading!

DayTrader - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:8080/daytrader/

DAYTRADER
PERFORMANCE BENCHMARKING

Home Trading & Portfolios Configuration Server Primitives FAQ About

DayTrader Home DayTrader

Home Account Portfolio Quotes Logout

Tue Jun 05 13:10:03 EDT 2007

Alert: The following Order(s) have completed.

order ID	order status	creation date	completion date	txn fee	type	symbol	quantity
0	completed	2007-06-05 09:49:02.875	2007-06-05 09:49:03.14	24.95	buy	s:26	41.0
order ID	order status	creation date	completion date	txn fee	type	symbol	quantity
1	completed	2007-06-05 09:49:03.218	2007-06-05 09:49:03.218	24.95	buy	s:374	139.0
order ID	order status	creation date	completion date	txn fee	type	symbol	quantity
2	completed	2007-06-05 09:49:03.265	2007-06-05 09:49:03.265	24.95	buy	s:4	146.0

Welcome uid:0,

User Statistics

account ID: 0
 account created: 2007-06-05 09:49:02.812
 total logins: 1
 session created: Tue Jun 05 13:09:54 EDT 2007

Account Summary

cash balance: \$971846.15
 number of holdings: 3
 total of holdings: \$28079.00
 sum of cash/holdings: \$999925.15
 opening balance: \$1000000.00

current gain/(loss): \$ -74.85 ↓ (+0.00%) ↑

Market Summary 2007-06-05

DayTrader Stock Index (TSIA) 102.08 (+4.00%) ↑

Trading Volume 23091.0

Top Gainers

symbol	price	change
s:171	297.54	100.54 ↑
s:156	283.50	98.50 ↑
s:187	208.71	85.71 ↑
s:139	253.51	76.51 ↑
s:114	196.22	59.22 ↑

Top Losers

symbol	price	change
s:160	95.12	-53.88 ↓
s:135	122.18	-42.82 ↓
s:109	146.44	-36.56 ↓
s:118	126.49	-36.51 ↓
s:147	136.24	-32.76 ↓

Note: Click any [symbol](#) for a quote or to trade.

quotes s:0, s:1, s:2, s:3, s:4

DayTrader Home DayTrader

GDF ↓ .15 HJK ↑ 1.25 RTY ↓ 1.23 IOP ↑ .05 BNM ↑ 12.0 XCV ↑ .20 QEW ↓ .65

Copyright 2006, Apache Software Foundation. All Rights Reserved

POWERED BY APACHE GERONIMO

Done

Additional details for configuring and running Daytrader can be found in the application **FAQ** available by pointing your web browser to <http://localhost:8080/daytrader>

Back to square one

After you performed some tests and want to run a new set from scratch you will need to reset the runtime configuration and transaction data from the database.

- Point your browser to <http://localhost:8080/daytrader>
- Click on **Configuration**.
- Click on **Reset DayTrader (to be done before each run)**.

- Click on **(Re)-populate DayTrader Database**.

These simple steps are all you need to start a new set of tests on Daytrader however, you may still want to restart the server depending on the type of tests you are running.

Launching the application clients

DayTrader provides two J2EE application clients, the DayTrader Streamer and a web services application. The Streamer application client uses a JMS topic to subscribe to quote price updates as stocks are bought and sold. These updates are tracked and used to determine if database collisions occur while updating the quote prices in the database. The web services application client simply provides a thick client for accessing DayTrader services using a web services interface.

Streamer application client

In order for the quote price updates to get published to the JMS topic, the "Publish Quote Updates" flag on the configuration page must be enabled.

- Point your browser to <http://localhost:8080/daytrader>
- Click on **Configuration**.
- Click on **Configure DayTrader run-time parameters**.
- Make sure you select the **Publish Quote Updates** checkbox.

To start the Streamer application client run the following command.

```
<geronimo_home>/bin/java -jar client.jar geronimo/daytrader-streamer-client/2.0-SNAPSHOT/car
```

The screenshot shows a window titled "Trade Client Application" with a "Streamer" tab. It displays a table of stock data under the heading "DAY TRADER PERFORMANCE BENCHMARKING". The table has columns for Symbol, Company Name, Price, Audit Price, Volume, Audit Volume, Open, Low, High, Change, and Last Update. The data is sorted by symbol (s:0 to s:120). The "Change" column shows price changes in green (positive) or red (negative). The "Last Update" column shows the time as 5/29/07 2:11 PM.

Symbol	Company Name	Price	Audit Price	Volume	Audit Volume	Open	Low	High	Change	Last Update
s:0	S0 Incorporated	\$241.24	\$241.24	188.0	188.0	\$181.00	\$181.00	\$181.00	\$60.24	5/29/07 2:11 PM
s:1	S1 Incorporated	\$44.44	\$44.44	451.0	451.0	\$42.00	\$42.00	\$42.00	\$2.44	5/29/07 2:11 PM
s:10	S10 Incorporated	\$120.00	\$120.00	0.0	0.0	\$120.00	\$120.00	\$120.00	\$0.00	5/29/07 2:11 PM
s:100	S100 Incorpora...	\$132.73	\$132.73	382.0	382.0	\$161.00	\$161.00	\$161.00	(\$28.27)	5/29/07 2:11 PM
s:101	S101 Incorpora...	\$27.60	\$27.60	137.0	137.0	\$24.00	\$24.00	\$24.00	\$3.60	5/29/07 2:11 PM
s:102	S102 Incorpora...	\$160.00	\$160.00	0.0	0.0	\$160.00	\$160.00	\$160.00	\$0.00	5/29/07 2:11 PM
s:103	S103 Incorpora...	\$92.00	\$92.00	0.0	0.0	\$92.00	\$92.00	\$92.00	\$0.00	5/29/07 2:11 PM
s:104	S104 Incorpora...	\$170.26	\$170.26	267.0	267.0	\$184.00	\$184.00	\$184.00	(\$13.74)	5/29/07 2:11 PM
s:105	S105 Incorpora...	\$25.98	\$25.98	482.0	482.0	\$27.00	\$27.00	\$27.00	(\$1.02)	5/29/07 2:11 PM
s:106	S106 Incorpora...	\$106.57	\$106.57	33.0	33.0	\$120.00	\$120.00	\$120.00	(\$13.43)	5/29/07 2:11 PM
s:107	S107 Incorpora...	\$92.26	\$92.26	267.0	267.0	\$134.00	\$134.00	\$134.00	(\$41.74)	5/29/07 2:11 PM
s:108	S108 Incorpora...	\$32.35	\$32.35	474.0	474.0	\$25.00	\$25.00	\$25.00	\$7.35	5/29/07 2:11 PM
s:109	S109 Incorpora...	\$171.00	\$171.00	0.0	0.0	\$171.00	\$171.00	\$171.00	\$0.00	5/29/07 2:11 PM
s:11	S11 Incorporated	\$10.10	\$10.10	250.0	250.0	\$11.00	\$11.00	\$11.00	(\$0.90)	5/29/07 2:11 PM
s:110	S110 Incorpora...	\$168.11	\$168.11	218.0	218.0	\$191.00	\$191.00	\$191.00	(\$22.89)	5/29/07 2:11 PM
s:111	S111 Incorpora...	\$184.36	\$184.36	605.0	605.0	\$125.00	\$125.00	\$125.00	\$59.36	5/29/07 2:11 PM
s:112	S112 Incorpora...	\$48.28	\$48.28	525.0	525.0	\$45.00	\$45.00	\$45.00	\$3.28	5/29/07 2:11 PM
s:113	S113 Incorpora...	\$81.52	\$81.52	204.0	204.0	\$68.00	\$68.00	\$68.00	\$13.52	5/29/07 2:11 PM
s:114	S114 Incorpora...	\$196.62	\$196.62	8.0	8.0	\$174.00	\$174.00	\$174.00	\$22.62	5/29/07 2:11 PM
s:115	S115 Incorpora...	\$86.72	\$86.72	264.0	264.0	\$78.00	\$78.00	\$78.00	\$8.72	5/29/07 2:11 PM
s:116	S116 Incorpora...	\$122.36	\$122.36	234.0	234.0	\$133.00	\$133.00	\$133.00	(\$10.64)	5/29/07 2:11 PM
s:117	S117 Incorpora...	\$167.69	\$167.69	145.0	145.0	\$137.00	\$137.00	\$137.00	\$30.69	5/29/07 2:11 PM
s:118	S118 Incorpora...	\$106.38	\$106.38	396.0	396.0	\$112.00	\$112.00	\$112.00	(\$5.62)	5/29/07 2:11 PM
s:119	S119 Incorpora...	\$212.30	\$212.30	61.0	61.0	\$193.00	\$193.00	\$193.00	\$19.30	5/29/07 2:11 PM
s:12	S12 Incorporated	\$108.00	\$108.00	0.0	0.0	\$108.00	\$108.00	\$108.00	\$0.00	5/29/07 2:11 PM
s:120	S120 Incorpora...	\$193.88	\$193.88	230.0	230.0	\$151.00	\$151.00	\$151.00	\$42.88	5/29/07 2:11 PM

Web Services application client

```
<geronimo_home>/bin/java -jar client.jar geronimo/daytrader-wsapp-client/2.0-SNAPSHOT/car
```