Erewhon Demonstration System Reference

VERSION 2018

Contents

Introduction

The JADE Erewhon demonstration system is an Internet-enabled online purchasing and tendering application. It has been built to give you a good appreciation of JADE's features, including Web deployment, JADE smart client technology, and Web services. You can also look at the code to see how a JADE system is built.

This document provides you with all of the information you need to install and run the JADE Erewhon demonstration system. It also looks at how some of JADE's key features are used.

This system is more than just a demonstration of JADE's capabilities. It is a resource that you can draw upon when building your own JADE applications. You can also use it for your own JADE presentations in the public arena.

The demonstration system has been created to enable an imaginary company called Erewhon Investments Inc. to trade internationally over the Internet.

Erewhon Investments Inc. is an online business specializing in the sale of high-value antiques, luxury homes, and luxury holidays. With some of the world's wealthiest individuals as clients, Erewhon Investments caters to a steadily growing niche market at the very top end. The company is based in New Zealand, yet operates in a global market place, with suppliers and customers all around the world.

Erewhon Investments needed a Web-based merchandising system that could support their two real-time sale processes: retail purchases and a tendering process where clients can lodge date-constrained tenders to bid on an item.

The application had to have the reliability to accommodate the numerous Erewhon agents, each of whom has multiple items for sale, and provide facilities for these agents to access item and sale details from any location in the world, at any time of the day.

Added to this was the complication of different agents using varying commission rates – the system had to be able to define multiple commission rates per sale item category, with the ability for different rates to apply to different agents.

In terms of client user requirements, the system had to have a search capability, to enable clients to search the database and create a list of items filtered from some or all of the following criteria: region, category, tender or retail, and price range.

Having completed the search, the client then had to be able to step through the search results and view details about each of the items filtered. The details had to include the name, description, price details, and a photograph.

From there, the client had to be able to make a retail purchase or bid on a tender item. Both orders had to be added to the client's shopping cart for later confirmation. The shopping cart had to be able to maintain a current list of the items selected for client query, confirmation, or deletion. Confirmation of the shopping cart processes the transactions.

As both agents and clients had to be able to access the system from geographically widespread locations, the system had to provide both thin client and Web deployment options.

Part 1 Setting Up the Erewhon Demo System

Before you install the Erewhon Investments demonstration system, make sure you have installed your JADE system.

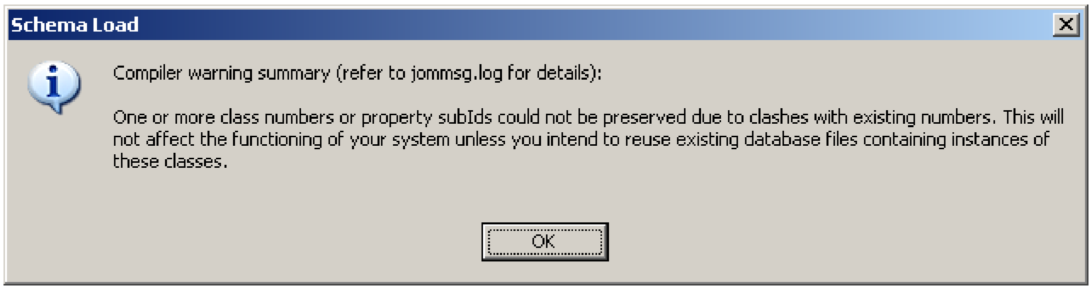
The following describes how to load the Erewhon schemas from the development environment. This requires the **examples\erewhon** folder to be local or visible via a file share from the machine on which you are running the development environment. If this is not the case, you can get the latest version of the Erewhon files from https://github.com/jadesoftwarenz/JADE-Erewhon, by following the instructions in the **README.md** Markdown language document. If you do not want to use the JADE development environment to load the **Erewhon** schemas, you can load them in batch mode. For details, see "Batch Loading the Erewhon Schemas", later in this document.

1. From the **Schema** menu, select **Load**.
2. In the Load Options dialog, check the **Load Multiple Schemas** check box.
3. Click the **Browse** button and find folder containing the **Erewhon** schema files. Select the **ErewhonInvestments.mul** file in this folder and then click **Open**.
4. In the Load Options dialog, click **OK** to load the demonstration system, or click **Cancel** to return to the main window.

   The following schemas are loaded into your environment.

   - **CommonSchema**
   - **ErewhonInvestmentsModelSchema**
   - **ErewhonInvestmentsViewSchema**
   - **SelfDocumentorSchema**
   - **WebServiceConsumer**.

   If a message box about class numbers and property subIds (shown in the following image) is displayed, click **OK** to ignore it.



Schema Load

Compiler warning summary (refer to jommsg.log for details):

One or more class numbers or property subIds could not be preserved due to clashes with existing numbers. This will not affect the functioning of your system unless you intend to reuse existing database files containing instances of these classes.

Batch Loading the Erewhon Schemas

The following describes how to batch-load the Erewhon schemas. Ensure that any JADE database and application servers for this system are shut down before proceeding.

1. Start a command line session on the host where your database is located.
2. Change to the ***<install-dir>*** folder, where *<install-dir>* is the folder in which your JADE system is installed.

3. Enter the following command, where the *Erewhon-dir* value is the folder containing the **Erewhon** schema files.

```
bin\jadloadb ini=<install-dir>\system\jade.ini path=<install-dir>\system
scmFile=Erewhon-dir\ErewhonInvestments.mul
```

Initializing the Erewhon Investments Database

The following describes how to initialize the Erewhon Investments database from the development environment. This requires the **\erewhon** folder to be local or visible via a file share from the machine on which you are running the development environment. If you want to initialize the database from the command line, see "Initializing the Database from the Command Line", later in this document.

1. In the Schema Browser window, select **ErewhonInvestmentsModelSchema**.
2. From the **Browse** menu, select **Classes**.
3. In the Class Browser, select the **JadeScript** class in the top-left panel of the window.
4. In the top-right pane of the window, scroll down the list of methods and then select the **initializeData** method.
5. From the **Jade** menu, select **Execute It** (or alternatively, press F9). This will run the method.
6. In the Browse for Folder form, find the **DataFiles** folder in the ***<install-dir>*\examples\erewhon** folder (where <*install-dir*> is the folder where you installed your JADE system). Select the **DataFiles** folder and then click **OK**. Click **Cancel** to return to the Class Browser window.
7. Progress messages are displayed in the Jade Interpreter Output Viewer Window during the database initialization. When the load completes, **Database initialized** is displayed in this window and **Execution complete** is displayed in the JADE status line. Select **Exit** from the **File** menu of the Jade Interpreter Output Viewer to close it.

The Erewhon demonstration system database is now initialized.

Initializing the Database from the Command Line

The following describes how to initialize the Erewhon Investments database from the command line. Ensure that any JADE database and application servers for this system are shut down before proceeding.

1. Start a command line session on the host where your database is located.
2. Change to the ***<install-dir>*** folder, where <*install-dir*> is the folder in which your JADE system is installed.
3. Enter the following command, where the *Erewhon-dir* value is the folder containing the **Erewhon** schema files.

```
bin\jadclient server=singleUser ini=<install-dir>\system\jade.ini
path=<install-dir>\system schema= ErewhonInvestmentsModelSchema
app=DataLoader startAppParameters Erewhon-dir\DataFiles
```

4. The database will now be initialized. You will see progress messages and when the load completes, a **Database initialized** message.

Running the Administration Application (Standard Client)

Select the Schema Browser window. If this window is not visible, select **Schema Browser** from the Window menu to set focus to it.

1. In the Schema Browser window, select **ErewhonInvestmentsViewSchema**.
2. Click the **Run Application** toolbar button (this is the one with the arrow icon).
3. Select **Administration** in the **Application Name** combo box. Click **OK** to start this application.
4. Select a name from the **User Name** combo box. The Company Administrator is **Erewhon Investments Inc.**
5. Click **OK** to run the application.

For information about using the application, see “Part 2 – User Guide”, later in this document.

Running the Shop Application (Standard Client)

1. Select the Schema Browser window. If this window is not visible, select **Schema Browser** from the Window menu to set focus to it.
2. In the Schema Browser window, select the **ErewhonInvestmentsViewSchema**.
3. Click the **Run Application** toolbar button (this is the one with the arrow icon).
4. Select **ErewhonShop** in the **Application Name** combo box. Click **OK** to run this application.
5. Select a name from the **User Name** combo box.
6. Click **OK** to run the application.

For information about using the application, see “Part 2 – User Guide”, later in this document.

Running the Tender Closure Application (Standard Client)

1. Select the Schema Browser window. If this window is not visible, select **Schema Browser** from the Window menu to set focus to it.
2. In the Schema Browser window, select **ErewhonInvestmentsViewSchema**.
3. Click the **Run Application** toolbar button (this is the one with the arrow icon).
4. Select **TenderClosureApp** in the **Application Name** combo box.
5. Click **OK** to run the application.

For information about using the application, see “Part 2 – User Guide”, later in this document.

Running JADE in Thin Client Mode

Note The JADE thin client communicates with the application server via TCP/IP. You must have TCP/IP installed and configured to use the JADE thin client.

To run JADE in thin client mode, you must create two shortcuts: one for the application server, and one for the thin client.

1. For the application server, create a shortcut with the following properties.

   Target:

   ```
   <install-dir>\bin\jadapp.exe
   path=<install-dir>\system
   server=singleUser
   appserverport=60000
   ini=<install-dir>\system\jade.ini
   ```

   Start In:

   ```
   <install-dir>\bin
   ```

   The <***install-dir***> value is the folder in which you installed your JADE system.

2. For the JADE thin client, create a shortcut with the following properties.

   Target:

   ```
   <install-dir>\bin\jade.exe
   schema=JadeSchema
   app=Jade
   appserver=<computer-name>
   appserverport=60000
   ```

   Start In:

   ```
   <install-dir>\bin
   ```

   The <***install-dir***> value is the folder in which you installed your JADE system and <***computer-name***> is the name or IP address of your computer (or **localhost**, or the loop-back IP address 127.0.0.1). To find your computer name or IP address, open a the **Network and Sharing Center** in Control Panel and then click **Local Area Connection** to view the IP address.

3. You can now run JADE in thin client mode.

   Note Ensure that you have shut down any open JADE sessions before running the application server. This includes the JADE development environment.

4. Start the application server from the application server shortcut. The application server will start and is now waiting for thin client connections.

5. Run JADE as a thin client of the application server from the thin client shortcut. The connection is displayed in the application server window. Log on to JADE as usual, and initiate applications using the standard client instructions in an earlier section. Notice that the interface presented under the JADE thin client is identical to the JADE standard client.

For information about using the application, see "Part 2 – User Guide", later in this document.

Running the Web Shop Application using Apache HTTP Server

See the *JADE Installation and Configuration Guide* (which is also available from the JADE Web site at https://www.jadeworld.com/developer-center/resource-library) for information about deploying JADE Web applications.

1. Before running the Web shop application, you must install the JADE HTTP driver for Apache, and define a virtual directory for JADE in your Apache configuration files. Copy the Windows version of the **mod_jadehttp.so** file to the Apache modules folder. Now edit the **conf/httpd.conf** file and at the end of the **Dynamic Shared Object (DSO) Support** section, add the following lines.

   ```
   LoadModule jadehttp_module modules/mod_jadehttp.so
   <IfModule mod_jadehttp.c>
       Include conf/jadehttp.conf
   </IfModule>
   ```

   Now create a file called **conf/jadehttp.conf** with the following details.

   ```
   <IfModule mod_jadehttp.c>
       <Location /jade-info>
           SetHandler jadehttp-info
       </Location>
       <Location /JadeEval>
           SetHandler jadehttp-handler
           Application WebShop
           TcpConnection 127.0.0.1 6107
       </Location>
   </IfModule>

   <Directory>
       require from all
   </Directory>
   Alias /images "C:\Temp"
   ```

   Create the **C:\Temp** folder on your machine if it does not already exist, and then restart your Apache HTTP Server.

   Note Ensure that you have shut down any open JADE sessions before proceeding further. This includes the JADE development environment and the JADE application server.

2. Click the **JADE** icon from your JADE program folder in the Start menu. This will run the JADE development environment in single-user mode.

3. Log on, and select **ErewhonInvestmentsViewSchema** in the Schema Browser window.

4. Click the **Run Application** toolbar button (this is the one with the arrow icon).

5. Select **WebShop** in the **Application Name** combo box.

6. Click **OK** to run the application.

7. The **WebShop** application will start and will appear as a single window. The application is now waiting for users to connect from a browser. If a warning message is displayed, advising you that you are running a Web application with an invalid Web working directory, you should click the **No** button on the message box, create the **C:\Temp** folder on your machine as stated in step 1 of this instruction, and then try again.

8. Bring up your Internet browser and then enter the following URL.

   ```
   <computer-name>/JadeEval?WebShop
   ```

   The <***computer-name***> value is the name or IP address of your machine (or **localhost** or the loop-back IP address 127.0.0.1), and **JadeEval** is the name of the virtual directory you created in your Web server; for example, a URL might be one of the following.

   - erewhon/JadeEval?WebShop
   - 192.168.1.100/JadeEval?WebShop
   - localhost/JadeEval?WebShop
   - 127.0.0.1/JadeEval?WebShop

Running the Web Shop Using Internet Information Server

See the *JADE Installation and Configuration Guide* (which is also available from the JADE Web site at https://www.jadeworld.com/developer-center/resource-library) for information about deploying JADE Web applications.

The following subsections contain instructions for running the **WebShop** application using Microsoft Internet Information Server (IIS).

- Configuring IIS
- Running the WebShop application
- Authorizing the WebShop application

Configuring IIS

The configuration instructions are grouped into the following subsections.

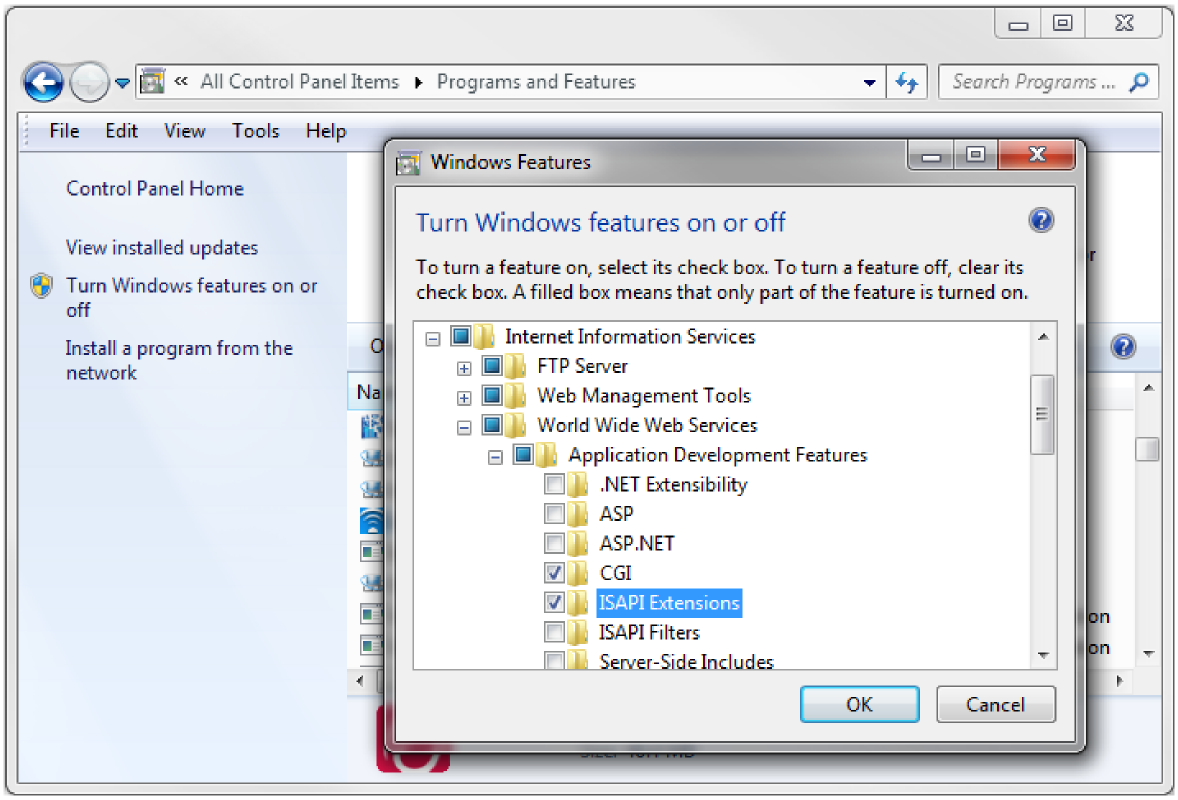
- Check that important IIS components are installed
- Add and configure an application (and application pool) for the **WebShop** application
- Add a virtual directory for **WebShop** image files

Step 1: Installing CGI and ISAPI Extensions

To install these optional components of IIS:

1. Select **Programs and Features** from the Control Panel.
2. Click the **Turn Windows features on or off** hyperlink on the left.
3. Expand **Internet Information Services**, then **World Wide Web Services**, and then **Application Development Features**.

4. Check the boxes **CGI** and **ISAPI Extensions** and then click the **OK** button.



Windows Features

Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

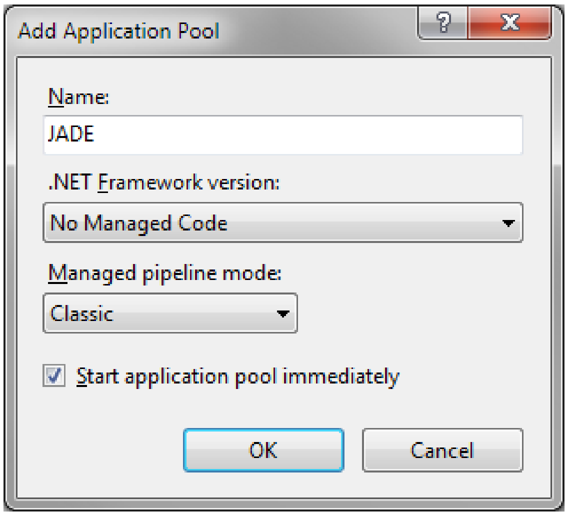
- Internet Information Services
  - FTP Server
  - Web Management Tools
  - World Wide Web Services
    - Application Development Features
      - [ ] .NET Extensibility
      - [ ] ASP
      - [ ] ASP.NET
      - [✓] CGI
      - [✓] ISAPI Extensions
      - [ ] ISAPI Filters
      - [ ] Server-Side Includes

Step 2: Adding an Application Pool

To add an application pool to be used by JADE Web applications:

1. Select **Administrative Tools** from the Control Panel.
2. Open **Internet Information Services (IIS) Manager**.
3. In the **Connections** panel on the left, select **Application Pools**.
4. Right-click and then select **Add Application Pool**.

5. Configure the pool to use unmanaged (non-.NET) code and then set the **Managed pipeline mode** field to **Classic**, as shown in the following image.



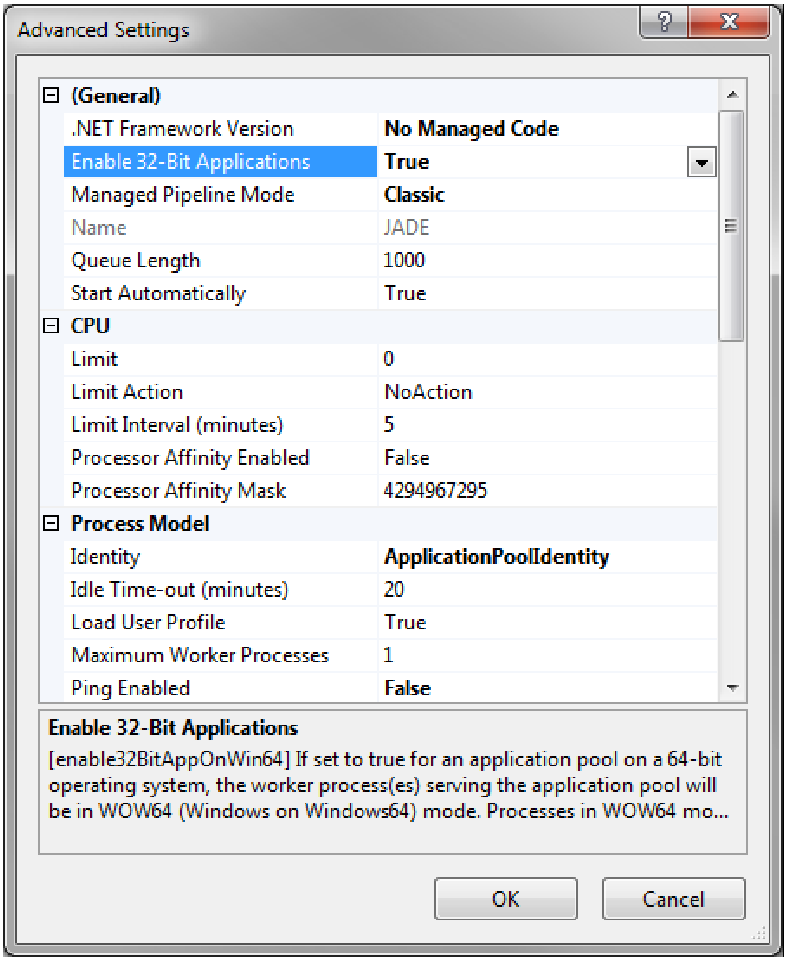
Name: JADE

.NET Framework version: No Managed Code

Managed pipeline mode: Classic

☑ Start application pool immediately

6. If you are using a 32-bit version of JADE, right-click on the application pool and then select **Advanced Settings**. In the Advanced Settings dialog, set **Enable 32-Bit Applications** to **True**.



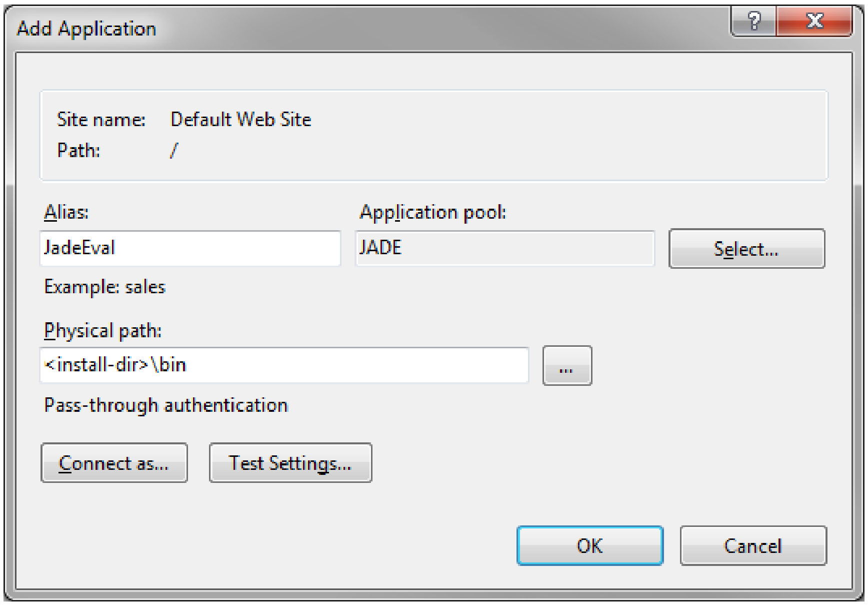
(General)	
.NET Framework Version	**No Managed Code**
Enable 32-Bit Applications	**True**
Managed Pipeline Mode	**Classic**
Name	JADE
Queue Length	1000
Start Automatically	True
CPU	
Limit	0
Limit Action	NoAction
Limit Interval (minutes)	5
Processor Affinity Enabled	False
Processor Affinity Mask	4294967295
Process Model	
Identity	**ApplicationPoolIdentity**
Idle Time-out (minutes)	20
Load User Profile	True
Maximum Worker Processes	1
Ping Enabled	**False**

Enable 32-Bit Applications
[enable32BitAppOnWin64] If set to true for an application pool on a 64-bit operating system, the worker process(es) serving the application pool will be in WOW64 (Windows on Windows64) mode. Processes in WOW64 mo...

Step 3: Adding an Application

To add an application:

1. Select the **Default Web Site** in the **Connections** panel.
2. Right-click and then select **Add Application**.
3. Enter the alias **JadeEval**. (This will be part of the URL for the **WebShop** application.)
4. Select the application pool that you created previously.
5. Enter the location of the **Physical path**, which is the **bin** folder for your JADE release; that is, replace **<install-dir>** in the following image with the correct location.


Add Application

Site name: Default Web Site
Path: /

Alias: JadeEval
Example: sales
Application pool: JADE

Physical path: <install-dir>\bin
Pass-through authentication

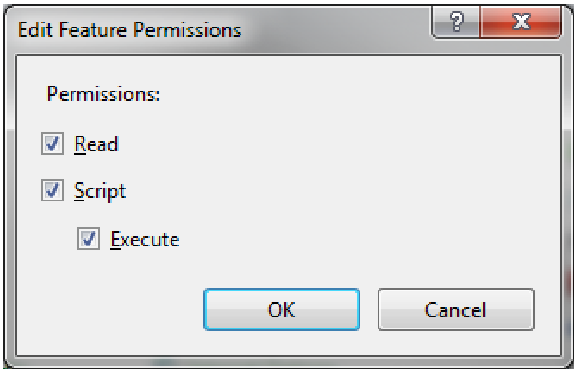
6. Click the **OK** button.

Step 4: Configuring Handler Mappings for the Application

To configure handler mappings for the application:

1. Select the application in the **Connections** panel.
2. Double-click the **Handler Mappings** icon in the central panel.
3. Right-click the **CGI-exe** handler mapping and select **Edit Feature Permissions**.

4. Enable all options, as shown in the following image.



Edit Feature Permissions

Permissions:

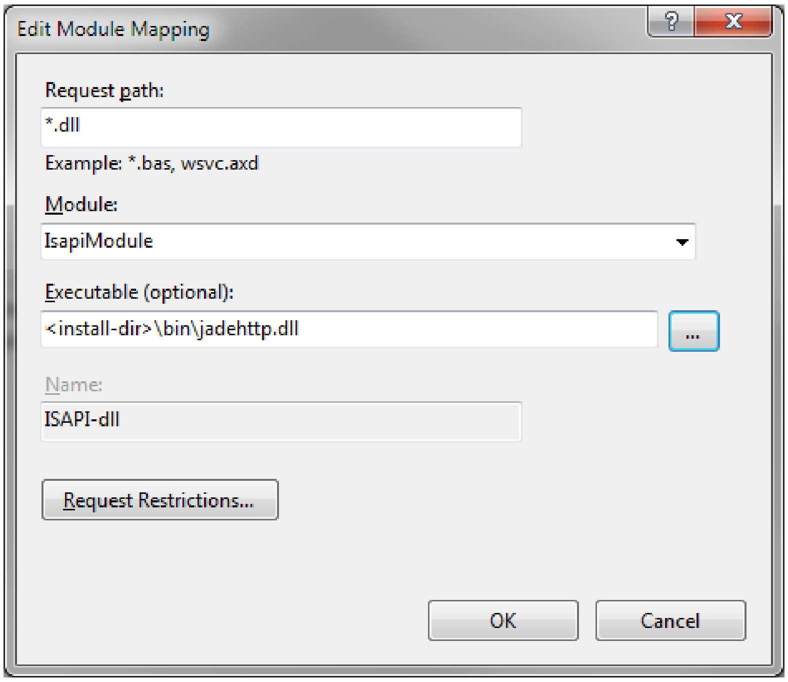
☑ Read

☑ Script

☑ Execute

5. Right-click the **ISAPI-dll** handler mapping and then select **Edit**.

6. Set the **Executable** text box to the path and file name of the **jadehttp.dll** file in the **bin** folder of your JADE system.



Edit Module Mapping

Request path: *.dll

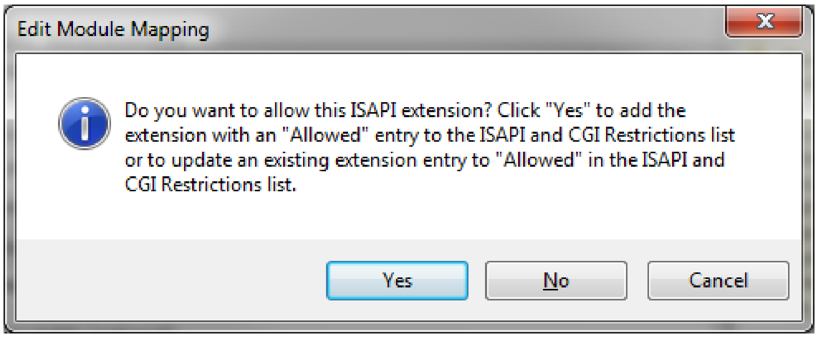
Example: *.bas, wsvc.axd

Module: IsapiModule

Executable (optional): <install-dir>\bin\jadehttp.dll

Name: ISAPI-dll

7. If the following dialog is displayed, click the **Yes** button.



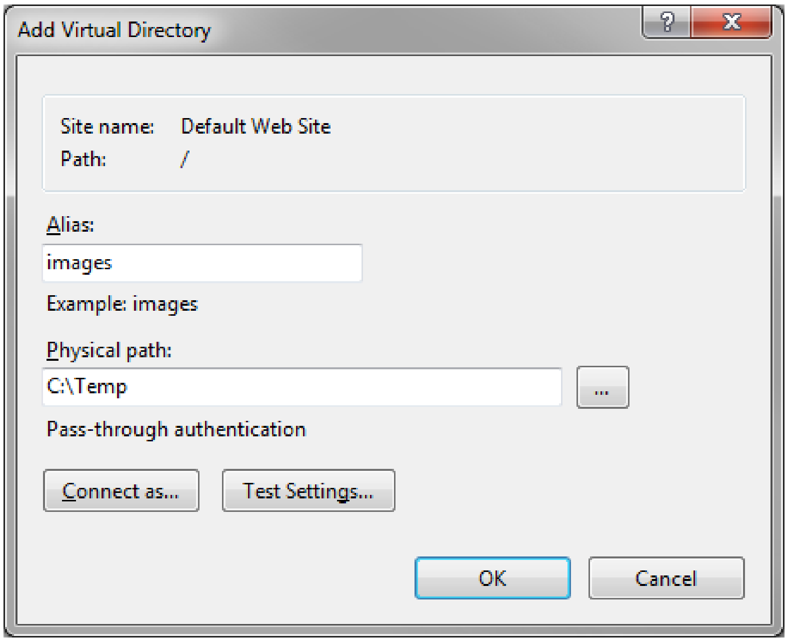
Edit Module Mapping

Do you want to allow this ISAPI extension? Click "Yes" to add the extension with an "Allowed" entry to the ISAPI and CGI Restrictions list or to update an existing extension entry to "Allowed" in the ISAPI and CGI Restrictions list.

Step 5: Adding a Virtual Directory for Images

To add a virtual directory for images:

1. Ensure that a **C:\Temp** folder exists on your machine.
2. Select **Default Web Site** in the **Connections** panel.
3. Right-click and then select **Add Virtual Directory**.
4. Complete the dialog as shown in the following image and then click the **OK** button.



Add Virtual Directory

Site name: Default Web Site
Path: /

Alias:
images
Example: images

Physical path:
C:\Temp

Pass-through authentication

5. Now start IIS for your Web site.

Note Ensure that you have shut down any open JADE sessions before proceeding further. This includes the JADE development environment and the JADE application server.

Running the Web Shop Application

With the IIS configuration completed, you can now attempt to run the **WebShop** application.

1. Click the **JADE** icon from your JADE program folder in the Start menu. This will run the JADE development environment in single-user mode.
2. Log on and in the Schema Browser window, select **ErewhonInvestmentsViewSchema**.
3. Click the **Run Application** toolbar button (this is the one with the arrow icon).
4. Select **WebShop** in the **Application Name** combo box.
5. Click **OK** to run the application, or **Cancel** to return to the main window.
6. The **WebShop** application will start and will appear as a single window. The application is now waiting for users to connect from a browser.
7. Open your Internet browser and then enter the following URL.

   ```
   <computer-name>/JadeEval/jadehttp.dll?WebShop
   ```

   The <***computer-name***> value is the name or IP address of your machine (or **localhost** or the loop-back IP address 127.0.0.1), and **JadeEval** is the name of the virtual directory you created in your Web server; for example, a URL could be one of the following.

   - erewhon/JadeEval/jadehttp.dll?WebShop
   - 192.168.1.100/JadeEval/jadehttp.dll?WebShop
   - localhost/JadeEval/jadehttp.dll?WebShop
   - 127.0.0.1/JadeEval/jadehttp.dll?WebShop
8. At this point, JADE's security features that protect against unauthorized running of applications via a Web browser will result in an error message (*Service unavailable* or similar) being displayed in your Web browser.

We must now authorize the **WebShop** application for IIS. For details, see the following section.

Authorizing the WebShop Application for IIS

We now need to specifically allow the **WebShop** application to be accessed via IIS from a Web browser.

Using Windows Explorer, you will find that JADE has created an additional four folders, as follows.

- <*install-dir*>\bin_jadehttp
- <*install-dir*>\bin_jadehttp\ini
- <*install-dir*>\bin_jadehttp\logs
- <*install-dir*>\bin_jadehttp\transfer

If these folders have not been created, ensure that IIS is running for your Web site. If it isn't, start it and then select the refresh option in your Web browser.

1. Using Notepad, open the **<*install-dir*>\bin_jadehttp\ini\jadehttp.ini** file.

   The <***install-dir***> value is the folder in which you installed your JADE system.

2. Add the following lines at the end of the file.

   ```
   [WebShop]
   TcpConnection=127.0.0.1
   TcpPort=6107
   ConnectionGroup=WebShopForms
   MinInUse=1
   MaxInUse=1
   CloseDelay=600
   ApplicationType=WebEnabledForms
   ```

   Save the file and then close Notepad.

3. Now return to your Web browser window and select the refresh option.

   You should now be presented with the **WebShop** logon form, at which point you can log on and run the application.

Part 2 User Guide

The JADE Erewhon Demo System consists of an **Administration** application (for Erewhon company and agent users) and a **Shop** application (for clients).

The **Administration** application is delivered as either a JADE standard client or a JADE thin client. The **Shop** application comprises two applications: one delivered as a JADE standard client or a JADE thin client, and the other as a JADE HTML thin client. For more details, see the following subsections.

Administration Application

The **Administration** application allows company and agent users to maintain core system data. For details, see the following subsections.

- Logon
- Main Administration Window - File Menu
- Main Administration Window - Edit Menu
- Main Administration Window - View Menu

Logon

To log on, select a user name from the drop-down list box on the Logon form then and click **OK**.

Note The list of user names includes the name of the **Erewhon Investments Inc.** company user and the names of all agents.

To access the **Administration** application's full range of functionality and data, select the **Erewhon Investments Inc.** user. Selecting an agent user name will mean that a limited subset of the **Administration** application's functionality and data will be available.

Main Administration Window – File Menu

To exit from the application, select the File menu **Exit** command (Alt+F, X), or press Alt+F4.

To view copyright and version details, select the File menu **About** command (Alt+F, A).

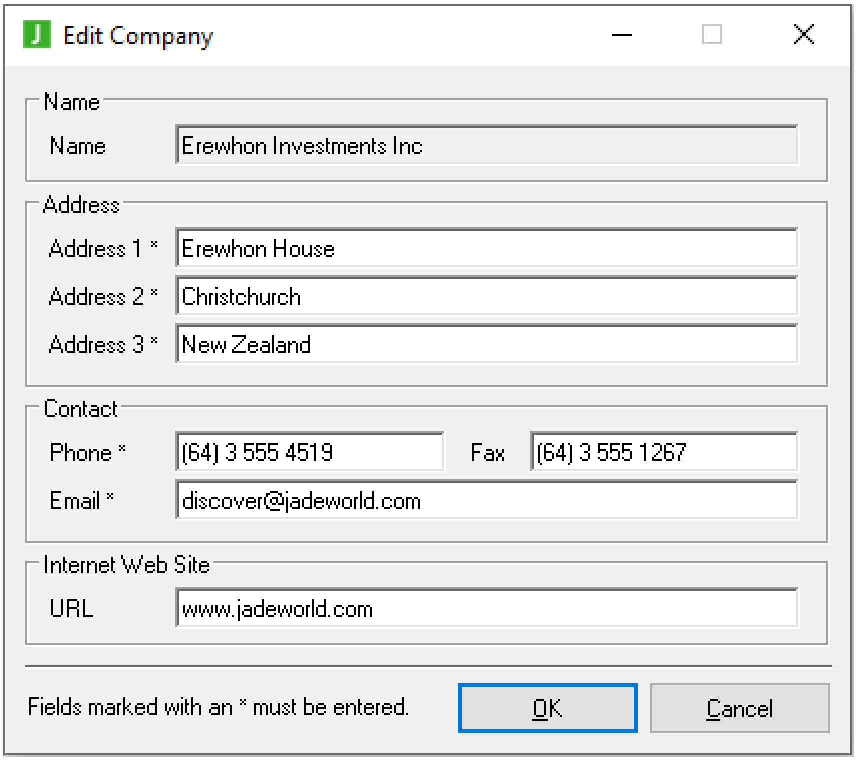
Main Administration Window – Edit Menu

This section contains the following topics.

- Company Details
- Agent Commission Rates (Company User Only)
- Locations (Company User Only)
- Sale Item Categories (Company User Only)

Company Details

Use the Edit Company screen, shown in the following image, to maintain company user details.



Name

Name: Erewhon Investments Inc

Address

Address 1 *: Erewhon House

Address 2 *: Christchurch

Address 3 *: New Zealand

Contact

Phone *: (64) 3 555 4519

Fax: (64) 3 555 1267

Email *: discover@jadeworld.com

Internet Web Site

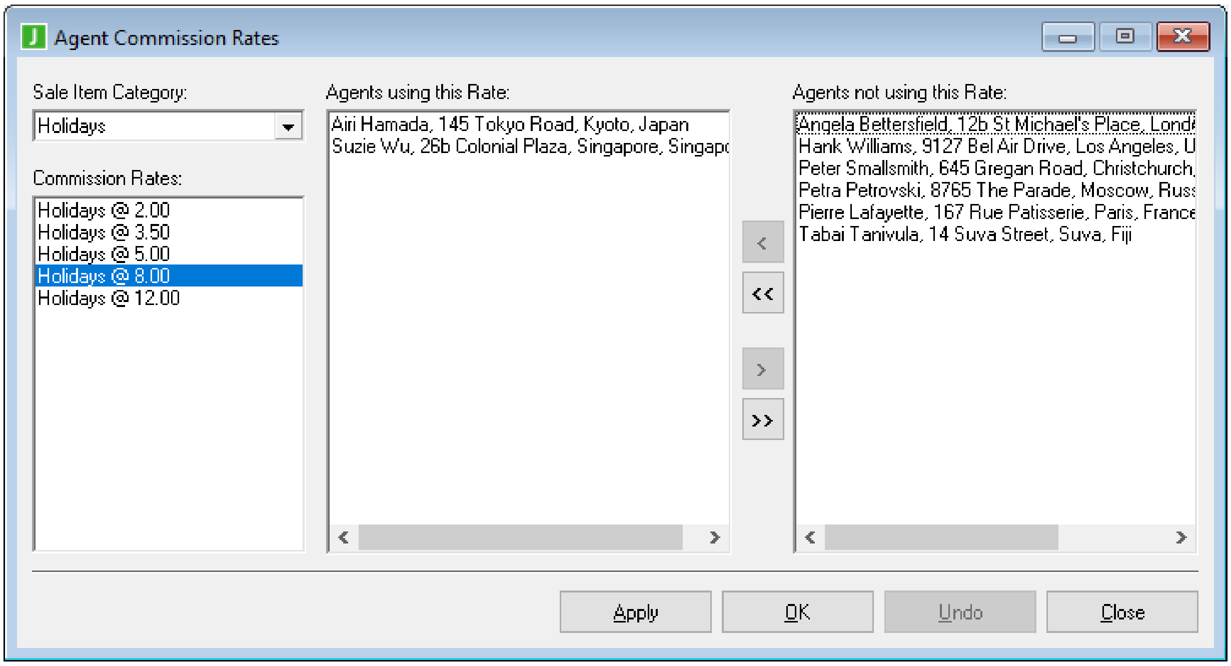
URL: www.jadeworld.com

Fields marked with an * must be entered.

To edit the company details, select **Edit Company Details** (Alt+E, D) from the Edit menu. Make changes as necessary and then click the **OK** button to update the database.

Agent Commission Rates (Company User Only)

Use the Agent Commission Rates screen, shown in the following image, to maintain a list of agents for each sale item's range of commission rates. (Note that an agent can use one rate only per sale item.)



Holidays @ 2.00
Holidays @ 3.50
Holidays @ 5.00
Holidays @ 8.00
Holidays @ 12.00

Airi Hamada, 145 Tokyo Road, Kyoto, Japan
Suzie Wu, 26b Colonial Plaza, Singapore, Singapo

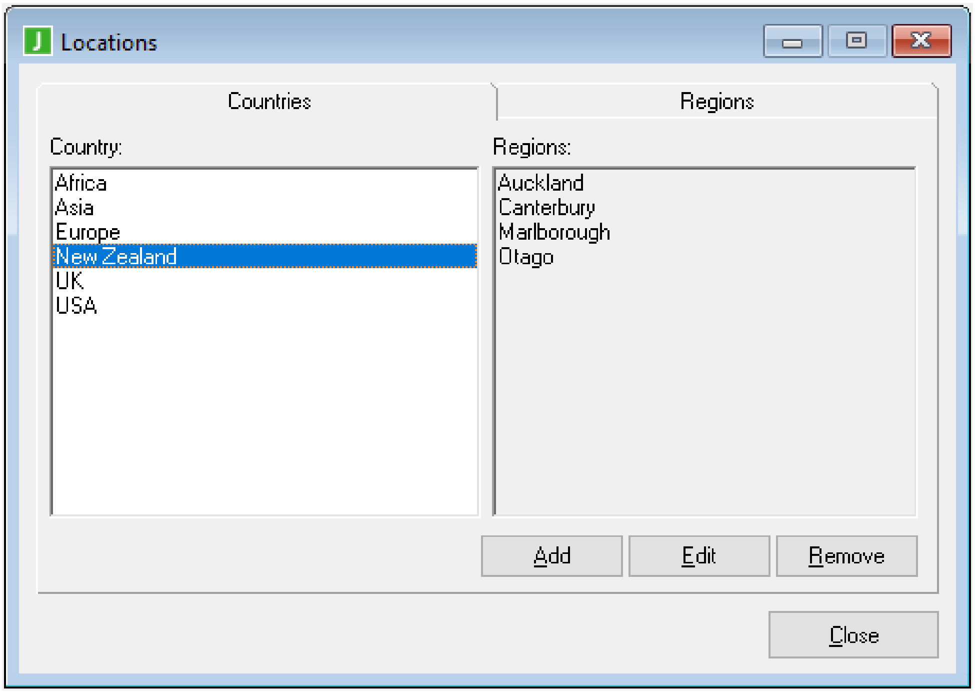
Angela Bettersfield, 12b St Michael's Place, Lond
Hank Williams, 9127 Bel Air Drive, Los Angeles, U
Peter Smallsmith, 645 Gregan Road, Christchurch,
Petra Petrovski, 8765 The Parade, Moscow, Rus
Pierre Lafayette, 167 Rue Patisserie, Paris, France
Tabai Tanivula, 14 Suva Street, Suva, Fiji

To modify the list of agents for a commission rate:

1. After opening the form, select a sale item category. The list of commission rates for that category is then displayed.
2. Select a commission rate from the list. This will populate two lists on the right: a list of all agents using the selected commission rate, and a list of all agents who do not use the selected commission rate.
3. To remove an agent from the list of those who use the selected commission rate, select the agent from the list on the left and then click the **>** button.
4. To remove all agents from the list of those who use the selected commission rate, click the **>>** button.
5. To add an agent to the list of those who use the selected commission rate, select the agent from the list on the right and then click the **<** button.
6. To add all agents to the list of those who use the selected commission rate, click the **<<** button.
7. Once you are satisfied with the list selections, click the **Apply** button to update the database or click the **OK** button to update the database *and* close the form. If you do not want to save the changes, click the **Undo** button.
8. To exit the form, without changing details, click **Close**.

Locations (Company User Only)

Use the Locations screen, shown in the following image, to maintain country and region details. Note that country is used to define a *macro* geographic area and region is used to define an associated *micro* geographic area. A form with a folder showing two tabs is then displayed. (Note that a region must always belong to a country.)

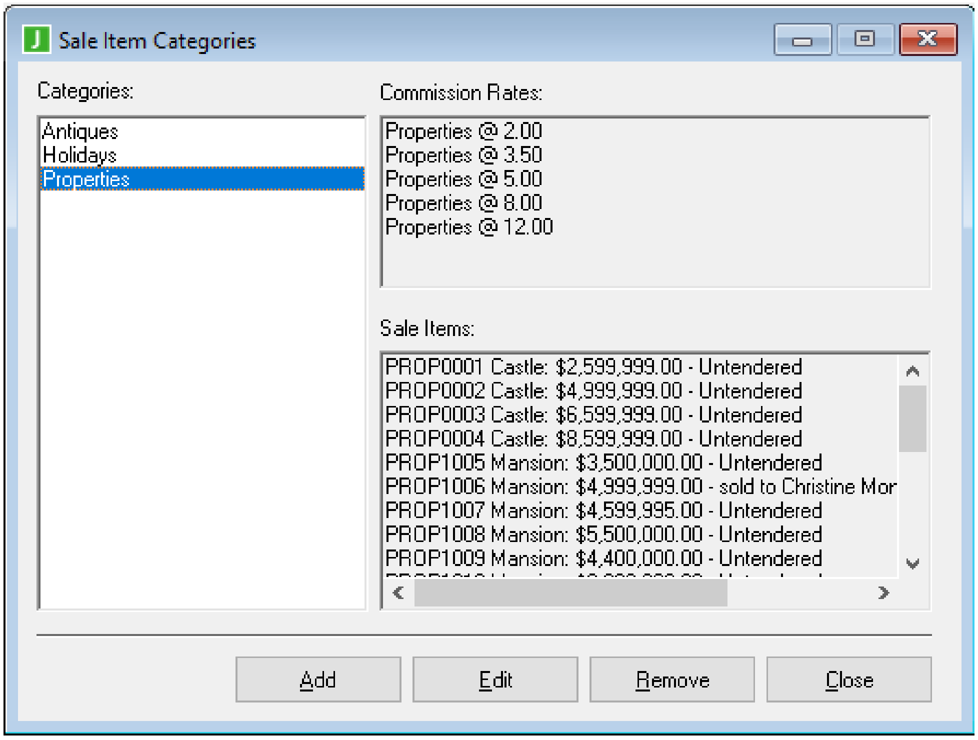


There are **Add**, **Edit**, and **Remove** buttons on both the **Countries** and **Regions** sheets.

- To add a new country or region, select the appropriate tab and then click the **Add** button. A form is then displayed, in which to enter the name of the location. Type the name and then click the **OK** button to add the location to the database.
- To change a country or region, select the appropriate tab and then click the **Edit** button. A form is then displayed, showing the current name. Change the name and then click the **OK** button to update the database.
- To remove a country or region, select the appropriate item and then click the **Remove** button. Note that when a country is removed, all of its regions are also removed.

Sale Item Categories (Company User Only)

Selecting the Edit menu **Edit Sale Item** command displays the Sale Items Categories form with a list of sale item categories. The right-hand panes display the sale item categories associated commission rates and sale items.



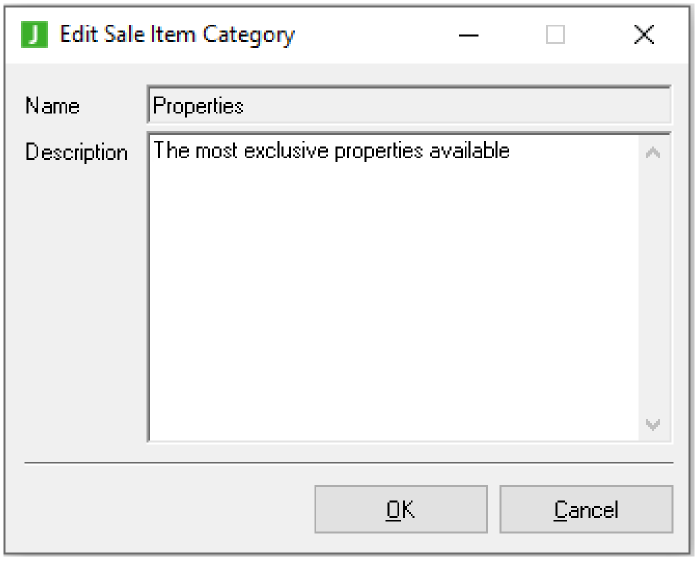
Antiques
Holidays
Properties

Properties @ 2.00
Properties @ 3.50
Properties @ 5.00
Properties @ 8.00
Properties @ 12.00

PROP0001 Castle: $2,599,999.00 - Untendered
PROP0002 Castle: $4,999,999.00 - Untendered
PROP0003 Castle: $6,599,999.00 - Untendered
PROP0004 Castle: $8,599,999.00 - Untendered
PROP1005 Mansion: $3,500,000.00 - Untendered
PROP1006 Mansion: $4,999,999.00 - sold to Christine Mor
PROP1007 Mansion: $4,599,995.00 - Untendered
PROP1008 Mansion: $5,500,000.00 - Untendered
PROP1009 Mansion: $4,400,000.00 - Untendered

- To add a new category, click the **Add** button. A form will then be displayed, in which to enter the name and description of the category. Type the name and description, then click the **OK** button to add the category to the database.

- To change a category, click the **Edit** button. A form will then be displayed, showing the current name and description. Change the description and then click the **OK** button to update the database (note that the name of the category cannot be changed).



Name: Properties

Description: The most exclusive properties available

- To remove a category, click the **Remove** button. Note that when a category is removed, all of its commission rates will also be removed.

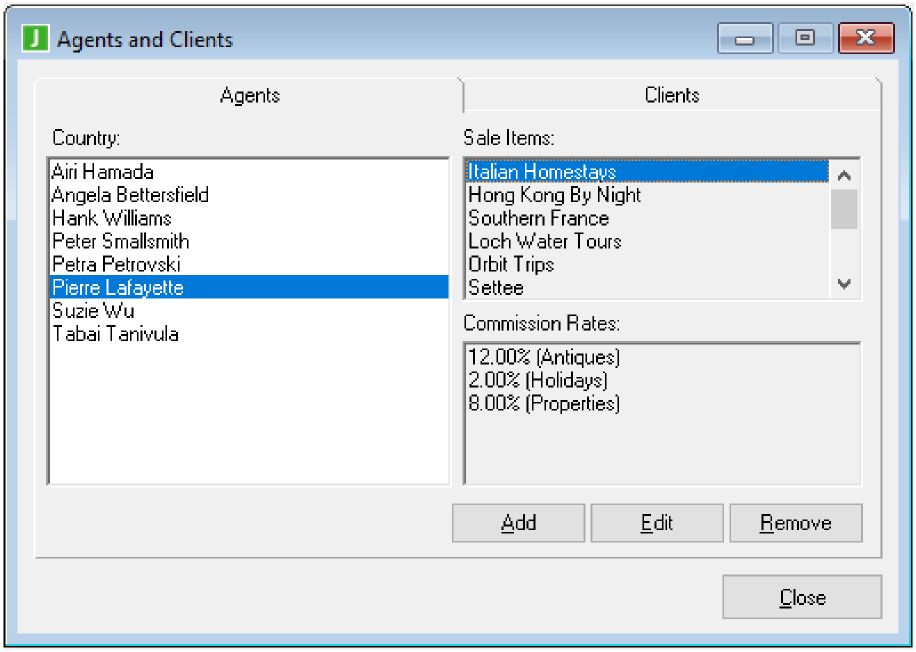
Main Administration Window – View Menu

This section contains the following topics.

- Agents and Clients (Company User Only)
- Commission Rates
- Sale Items (by Category)
- Sales

Agents and Clients (Company User Only)

Use the Agents and Clients screen to display a form with a folder containing two tabs: one for a list of agents and the other for a list of clients.



Airi Hamada
Angela Bettersfield
Hank Williams
Peter Smallsmith
Petra Petrovski
Pierre Lafayette
Suzie Wu
Tabai Tanivula

Italian Homestays
Hong Kong By Night
Southern France
Loch Water Tours
Orbit Trips
Settee

12.00% (Antiques)
2.00% (Holidays)
8.00% (Properties)

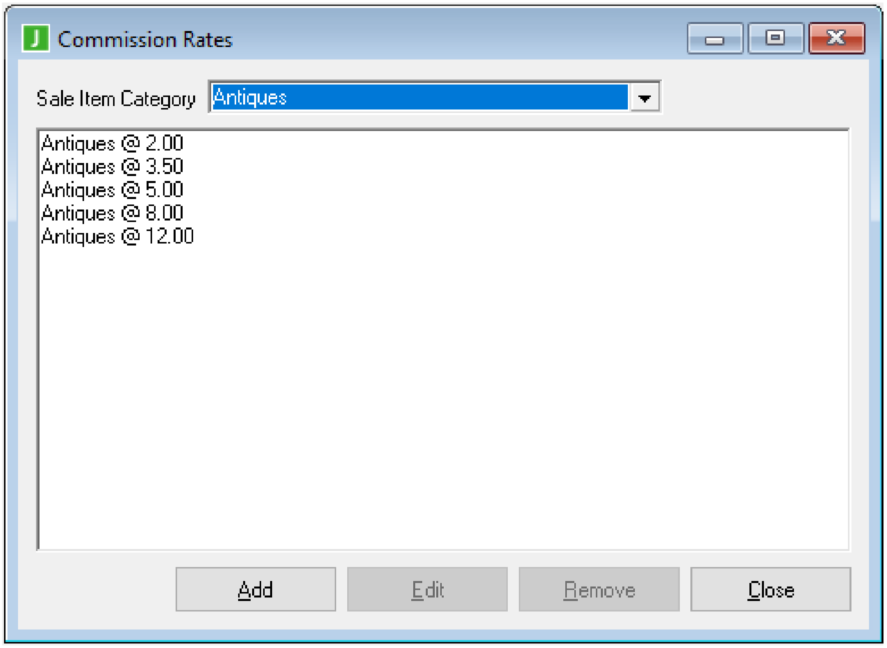
On each sheet there are **Add**, **Edit**, and **Remove** buttons.

- To add a new agent or client, select the required tab and then click the **Add** button. A form will be displayed, in which to enter the agent's or client's details. Enter the details and then click the **OK** button to add the agent or client to the database.
- To change an agent's or client's details, select the required tab and then click the **Edit** button. A form will be displayed, showing the details. Make any changes and then click the **OK** button to update the database.
- To remove an agent or client, select the required tab and then click the **Remove** button. Note that when a client is removed, all of its retail sales, tender sales, and outstanding tenders will also be removed.

Commission Rates

Use the Commission Rates screen to add or maintain variable commission rates for each sale item category.

A form displays with a drop-down list of sale item categories. Upon selecting a sale item category, a list is displayed of all commission rates for that category.



Antiques @ 2.00
Antiques @ 3.50
Antiques @ 5.00
Antiques @ 8.00
Antiques @ 12.00

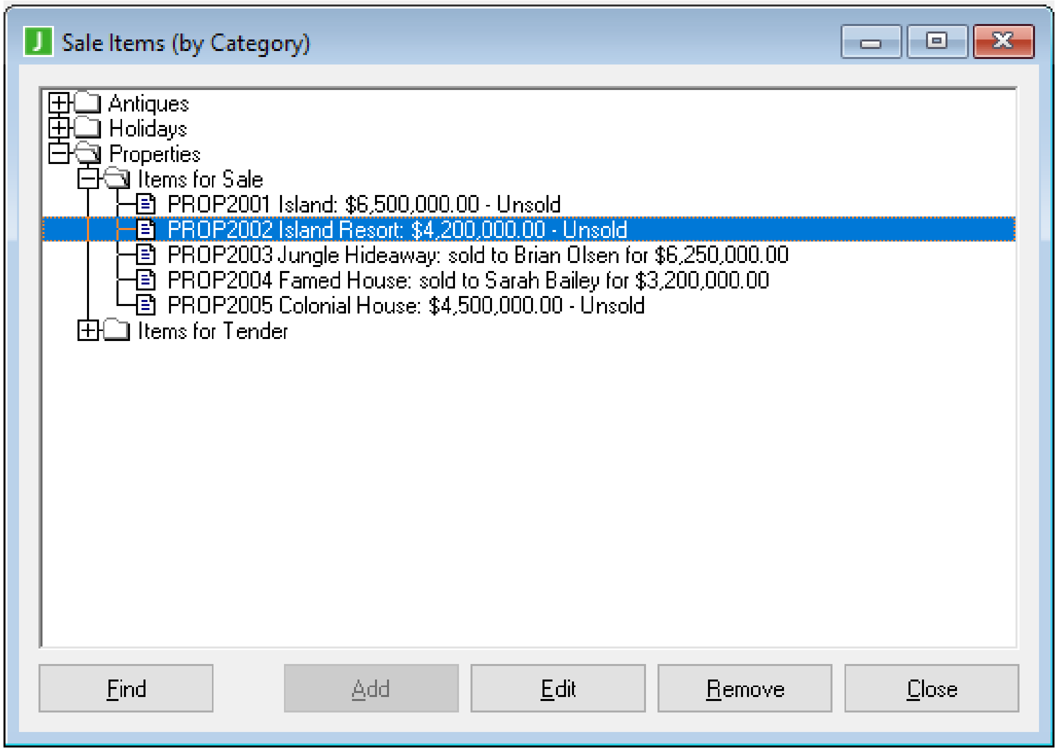
This form also has the following maintenance capabilities (company user only).

- To add a new commission rate, select your required sale item category from the drop-down list box and then click the **Add** button. A form will be displayed, in which to select the rate's category and enter the rate percentage. Enter the category and percentage, and then click the **OK** button to add the new commission rate to the database.
- To change a commission rate, select your required sale item category from the drop-down list box and the required commission rate and then click the **Edit** button. A form will be displayed, showing the current sale item category and rate percentage. Change the details to met your requirements and then click the **OK** button to update the database.
- To remove a commission rate, select it and then click the **Remove** button.

Sale Items (by Category)

Use the Sale Items (by Category) screen to add or maintain sale items. Note that only agents can add a new sale item. This screen employs a hierarchical tree of sale item categories, which can be expanded by clicking the **+** icon or collapsed by clicking the **-** icon.

As each sale item category's folder is opened, two subfolders are shown: one for the **Items for Sale**, and the other for the **Items for Tender**. Opening either of the subfolders will display a list of their respective sale items.


Sale Items (by Category)

Antiques
Holidays
Properties
Items for Sale
PROP2001 Island: $6,500,000.00 - Unsold
PROP2002 Island Resort: $4,200,000.00 - Unsold
PROP2003 Jungle Hideaway: sold to Brian Olsen for $6,250,000.00
PROP2004 Famed House: sold to Sarah Bailey for $3,200,000.00
PROP2005 Colonial House: $4,500,000.00 - Unsold
Items for Tender

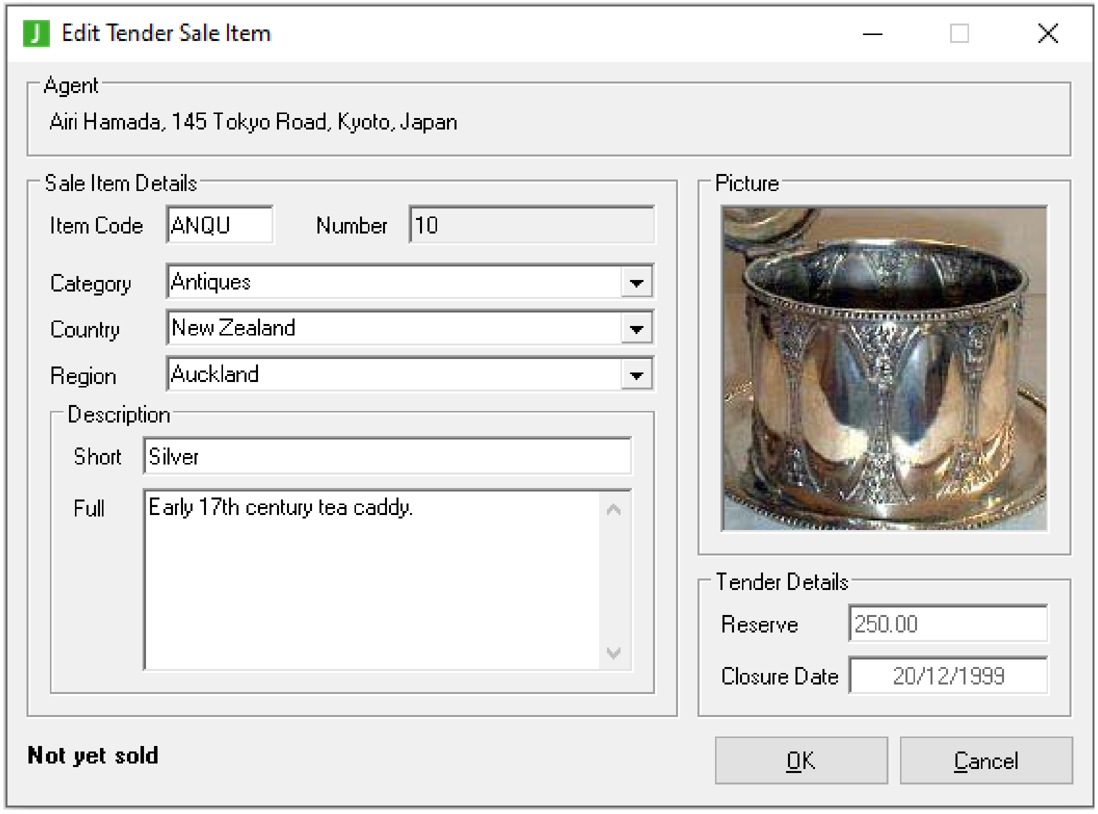
This form also has the following maintenance capabilities.

- To add a new sale item, open the folder of the sale item category to which the new sale item will belong, select the **Items for Sale** or **Items for Tender** subfolder, and then click the **Add** button (note that the **Add** button can also be clicked if a sale item leaf is selected).

  A form will be displayed, in which to enter the details of the sale item (the sale item category will have been preselected).

  To load a photo of the sale item from a disk file, double-click the empty picture frame. Enter the remainder of the details and then click the **OK** button to add the sale item to the database. Note that only agents can add sale items.

- To change a sale item, open the required sale item category folder and subfolder. Next, select the sale item that you want to change, and then click the **Edit** button. A form will be displayed, showing the current sale item details. Change the details to meet your requirements and then click the **OK** button to update the database.


Edit Tender Sale Item

Agent
Airi Hamada, 145 Tokyo Road, Kyoto, Japan

Sale Item Details
Item Code: ANQU
Number: 10
Category: Antiques
Country: New Zealand
Region: Auckland

Description
Short: Silver
Full: Early 17th century tea caddy.

Picture

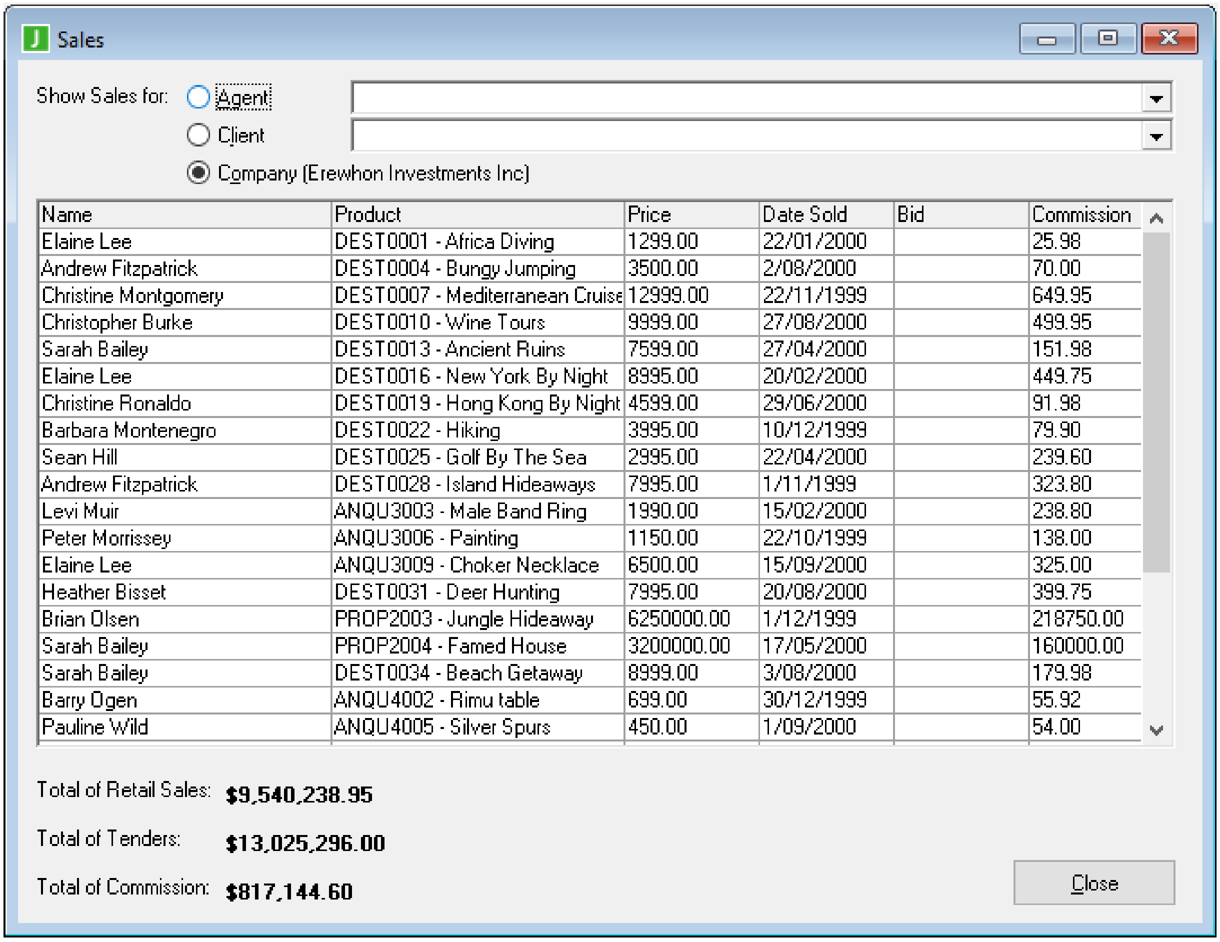
Tender Details
Reserve: 250.00
Closure Date: 20/12/1999

Not yet sold

- To remove a sale item, click the **Remove** button. Note that when a tender sale item is removed, all associated tenders are also removed.
- While a sale item can be added only by an agent, it can be edited or removed by that agent or by the company user.

Sales

Use the Sales screen to view sales summaries by agent, client, or for the whole company.



Sales

Show Sales for: Agent Client [✓ Company (Erewhon Investments Inc)]

Name	Product	Price	Date Sold	Bid	Commission
Elaine Lee	DEST0001 - Africa Diving	1299.00	22/01/2000		25.98
Andrew Fitzpatrick	DEST0004 - Bungy Jumping	3500.00	2/08/2000		70.00
Christine Montgomery	DEST0007 - Mediterranean Cruise	12999.00	22/11/1999		649.95
Christopher Burke	DEST0010 - Wine Tours	9999.00	27/08/2000		499.95
Sarah Bailey	DEST0013 - Ancient Ruins	7599.00	27/04/2000		151.98
Elaine Lee	DEST0016 - New York By Night	8995.00	20/02/2000		449.75
Christine Ronaldo	DEST0019 - Hong Kong By Night	4599.00	29/06/2000		91.98
Barbara Montenegro	DEST0022 - Hiking	3995.00	10/12/1999		79.90
Sean Hill	DEST0025 - Golf By The Sea	2995.00	22/04/2000		239.60
Andrew Fitzpatrick	DEST0028 - Island Hideaways	7995.00	1/11/1999		323.80
Levi Muir	ANQU3003 - Male Band Ring	1990.00	15/02/2000		238.80
Peter Morrissey	ANQU3006 - Painting	1150.00	22/10/1999		138.00
Elaine Lee	ANQU3009 - Choker Necklace	6500.00	15/09/2000		325.00
Heather Bisset	DEST0031 - Deer Hunting	7995.00	20/08/2000		399.75
Brian Olsen	PROP2003 - Jungle Hideaway	6250000.00	1/12/1999		218750.00
Sarah Bailey	PROP2004 - Famed House	3200000.00	17/05/2000		160000.00
Sarah Bailey	DEST0034 - Beach Getaway	8999.00	3/08/2000		179.98
Barry Ogen	ANQU4002 - Rimu table	699.00	30/12/1999		55.92
Pauline Wild	ANQU4005 - Silver Spurs	450.00	1/09/2000		54.00

Total of Retail Sales: **$9,540,238.95**

Total of Tenders: **$13,025,296.00**

Total of Commission: **$817,144.60**

To list the sales, select the agent, client, or company option using the radio buttons. If the agent or client option is selected, one of those entities must be selected from the drop-down list box to display the sales summary specific to that agent or client. To sort the results, click on any of the table column headers to sort by that column.

Notes When this summary is viewed by an agent user, only his or her sales appear in the list.

Total amounts are displayed at the bottom of the screen.

JADE Thin Client Shop Application

The JADE thin client shop application is the shop-front interface for clients to run over local or wide-area networks, or the Internet. The **ErewhonShop** application can also be run as a standard JADE client application (that is, a two-tier or *fat* client application). For details, see the following subsections.

- Logon
- Product Search
- Viewing the Details of a Product

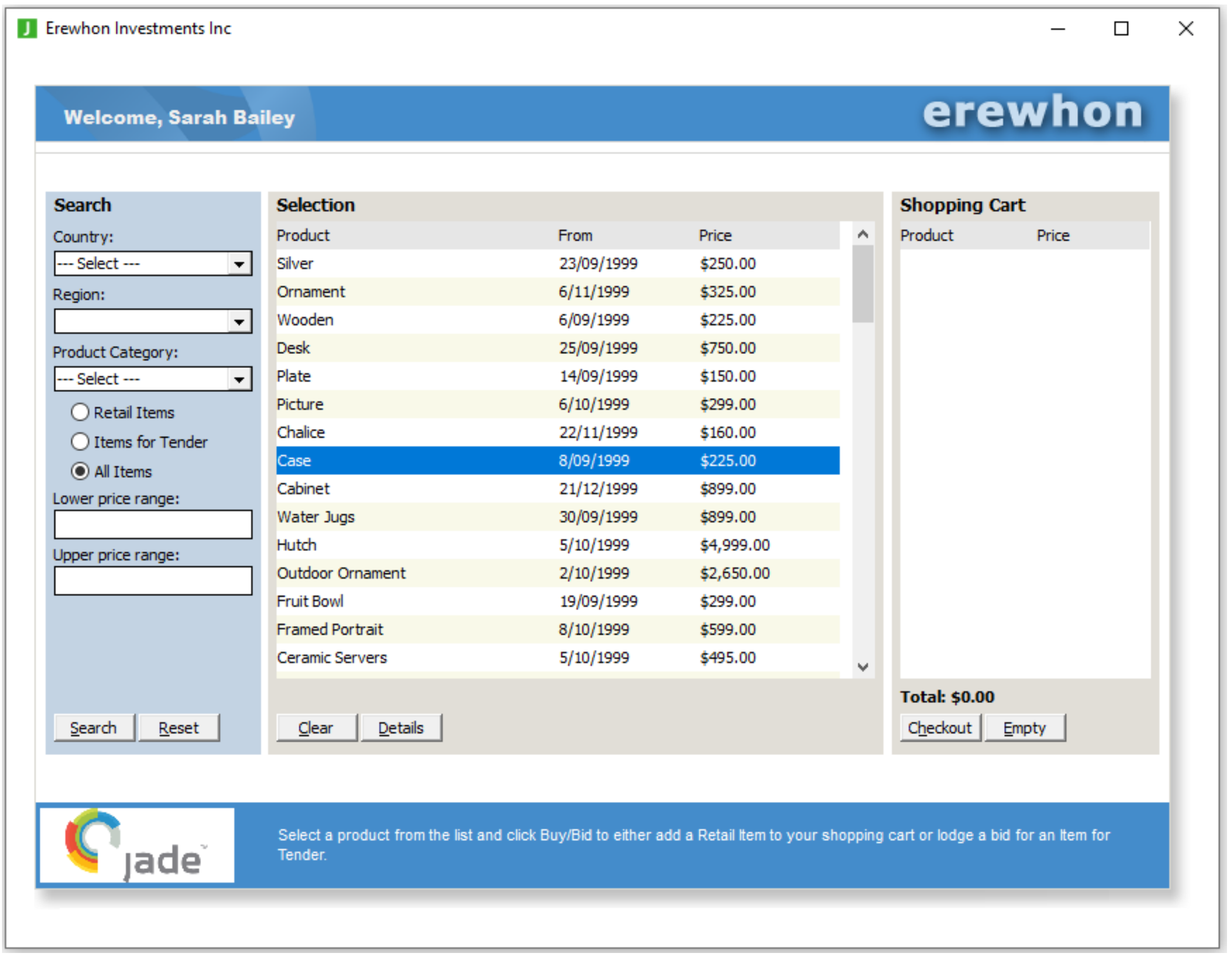
- Buying or Bidding for a Product
- Shopping Cart
- Product Details
- Checkout

Logon

On the logon form, select a user name from the drop-down list box and then click the **OK** button. To cancel the logon, click the **Cancel** button.

Product Search

The Product Search form is the main form of the **ErewhonShop** application. To begin, search for a list of products by selecting the required search criteria in the **Search** panel, and then click the **Search** button.



Erewhon Investments Inc

Welcome, Sarah Bailey

erewhon

Search

Country: --- Select ---

Region:

Product Category: --- Select ---

( ) Retail Items
( ) Items for Tender
(●) All Items

Lower price range:

Upper price range:

Search | Reset

Selection

Product	From	Price
Silver	23/09/1999	$250.00
Ornament	6/11/1999	$325.00
Wooden	6/09/1999	$225.00
Desk	25/09/1999	$750.00
Plate	14/09/1999	$150.00
Picture	6/10/1999	$299.00
Chalice	22/11/1999	$160.00
Case	8/09/1999	$225.00
Cabinet	21/12/1999	$899.00
Water Jugs	30/09/1999	$899.00
Hutch	5/10/1999	$4,999.00
Outdoor Ornament	2/10/1999	$2,650.00
Fruit Bowl	19/09/1999	$299.00
Framed Portrait	8/10/1999	$599.00
Ceramic Servers	5/10/1999	$495.00

Clear | Details

Shopping Cart

Product	Price

Total: $0.00

Checkout | Empty

jade

Select a product from the list and click Buy/Bid to either add a Retail Item to your shopping cart or lodge a bid for an Item for Tender.

A new search list can be generated at any time by changing the search criteria and then clicking the **Search** button again. Search criteria can also be reset by clicking the **Reset** button. The list of search results can be cleared at any time by clicking the **Clear** button below the list of results (this will not affect any of the items currently in the shopping cart).

Viewing the Details of a Product

To see more-detailed information about a product, select it in the search list and then click the **Details** button. This button will then be replaced by a button with a caption of **List** and the search results are replaced with details of the selected item. Clicking the **List** button while the item details are displayed returns to the list of search results.

Note If no product item is selected while viewing the list of search results, the **Details** button is disabled.

Buying or Bidding for a Product

Some products are retail sale items, while others are tender items for which you must enter a bid.

To buy a retail sale item, select it in the search results list and then click the **Buy/Bid** button, or click the **Buy/Bid** button while the item details for that item are displayed.

If the product is a tender item (that is, it requires you to make a bid), you must first click the **Details** button to see the item details for the item, and then enter a tender amount greater than or equal to the minimum price of the item.

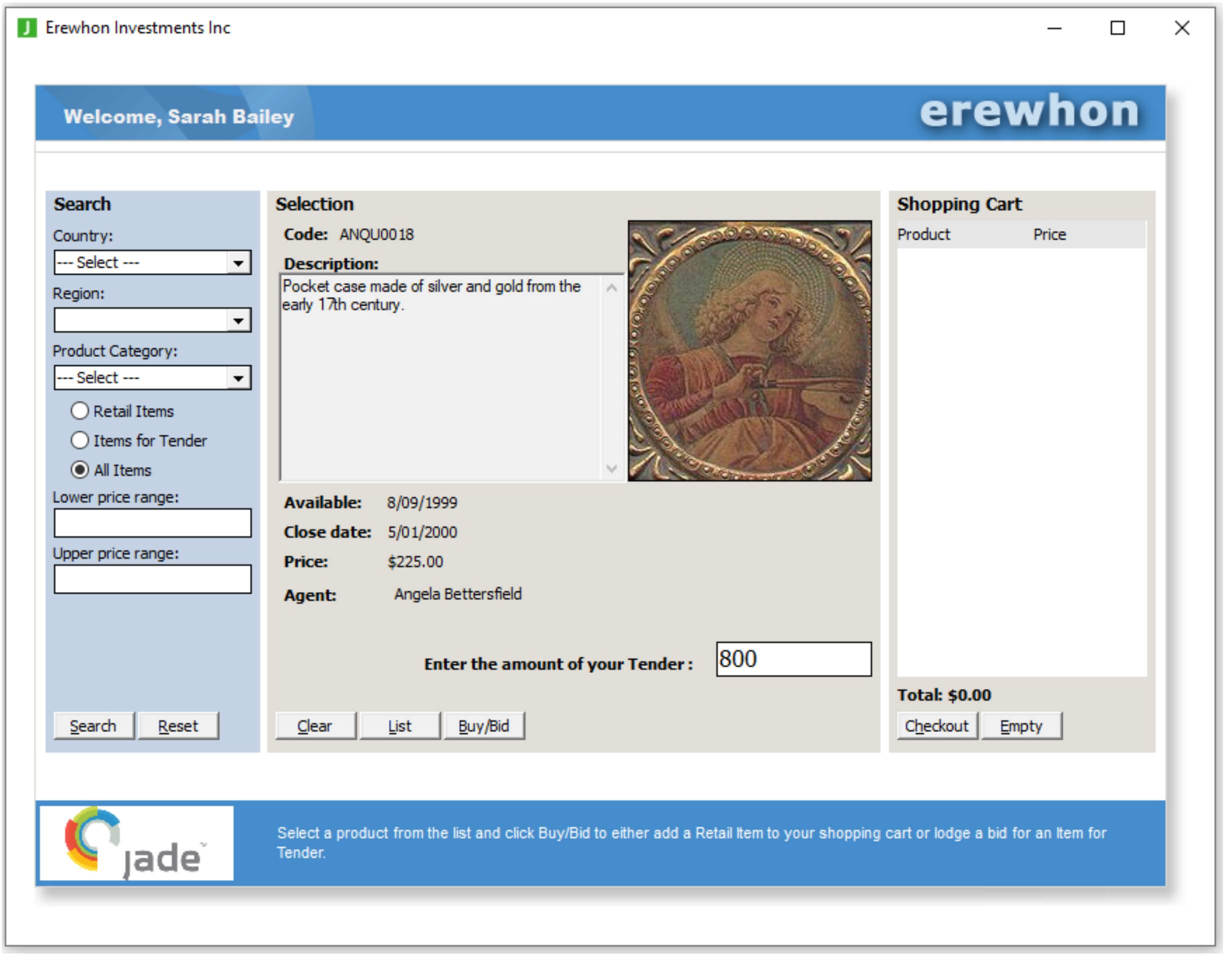
After entering your offer, click the **Buy/Bid** button again and the tender will then be added to your shopping cart. Alternatively, if you do not want to bid on the item, click the **List** button to return to the search results list.

Shopping Cart

The shopping cart list will be updated whenever a product item is bought or tendered for. A running total is also displayed beneath the cart list. To empty the shopping cart, click the **Empty** button. To go to the checkout, click the **Checkout** button.

Product Details

When a product is selected in the search results list, the **Details** button is enabled. Click this button to display more-detailed information about the product item.



Code: ANQU0018

Description: Pocket case made of silver and gold from the early 17th century.

Available: 8/09/1999

Close date: 5/01/2000

Price: $225.00

Agent: Angela Bettersfield

Enter the amount of your Tender : 800

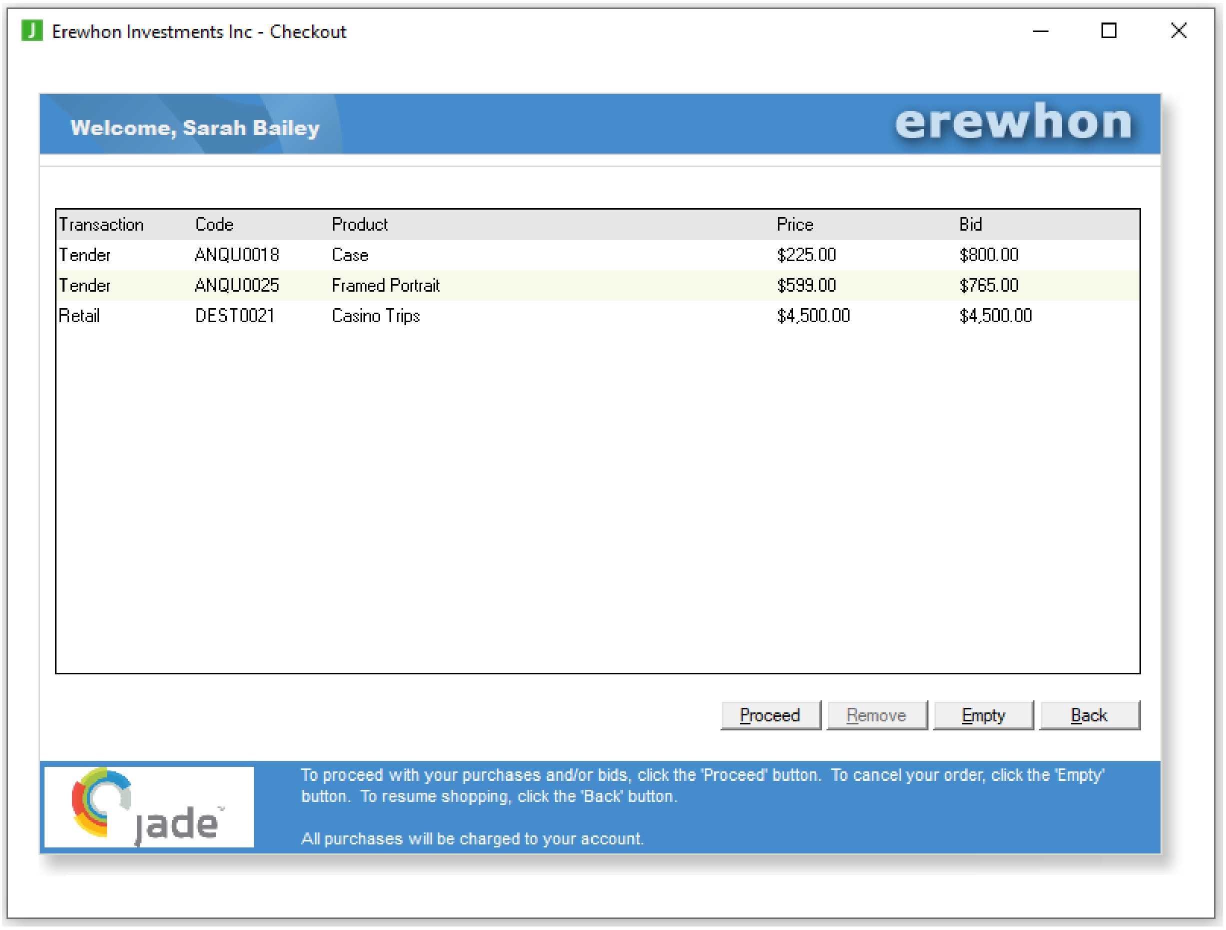
Select a product from the list and click Buy/Bid to either add a Retail Item to your shopping cart or lodge a bid for an Item for Tender.

If the product item is a tender item, the details of the product include a field in which to enter your offer (your tender amount). The same product searching and shopping cart functions are available as when the search results list is displayed (see above).

If the **Clear** button is clicked with the product details in view, the details will be replaced with an empty search results list. Clicking the **Reset** button (in the **Search** panel) with a product's details displayed will have the same effect as the **Clear** button, but the search criteria will also default to their original settings. If the **Search** button is clicked with a product's details displayed, the details will be replaced with the (new) search results list.

Checkout

The checkout is the final confirmation of your shopping cart before proceeding with the transaction of purchasing sale items or submitting any bids for tender items, or both purchasing sale items and submitting any bids for tender items.



Transaction	Code	Product	Price	Bid
Tender	ANQU0018	Case	$225.00	$800.00
Tender	ANQU0025	Framed Portrait	$599.00	$765.00
Retail	DEST0021	Casino Trips	$4,500.00	$4,500.00

To proceed with your purchases and/or bids, click the 'Proceed' button. To cancel your order, click the 'Empty' button. To resume shopping, click the 'Back' button.

All purchases will be charged to your account.

To remove any unwanted items from the shopping cart, select the item in the list and then click the **Remove** button. To remove all of the items in the shopping cart, click the **Empty** button. If you want to return to the Product Search form, click the **Back** button. To initiate the final processing of the shopping cart, click the **Proceed** button. A list of the bought and tendered items will then be displayed so that you can confirm the transaction.

Web Shop Application

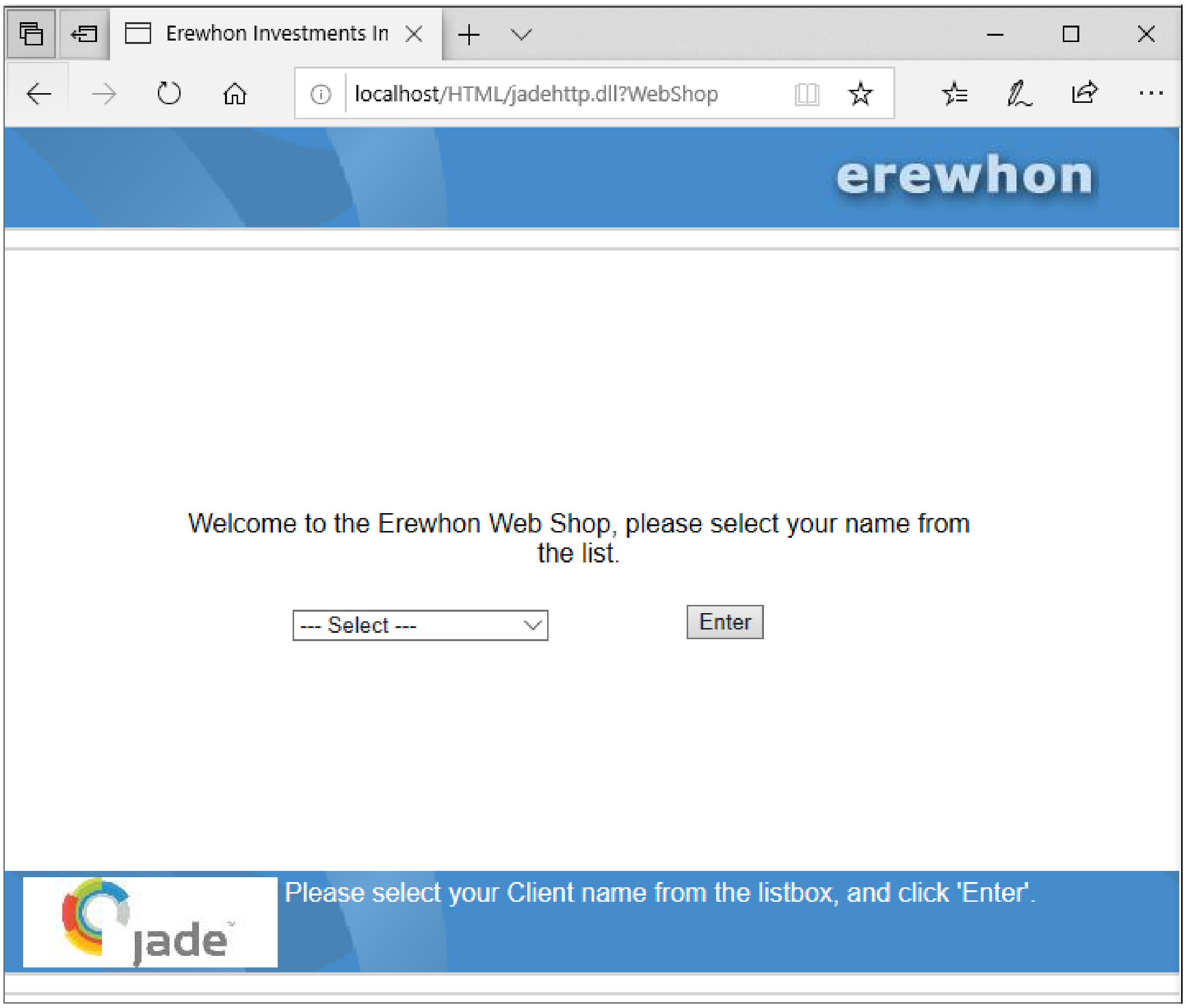
The **WebShop** application is the shop-front interface for clients to run over the World-Wide-Web. The application is built in JADE and is automatically deployed over the Internet using JADE's native functionality. For details, see the following subsections.

- Logon
- Product Search
- Viewing the Details of a Product
- Buying or Bidding for a Product

- Product Details - Tender
- Checkout

Logon

The following image is the **WebShop** application logon form.



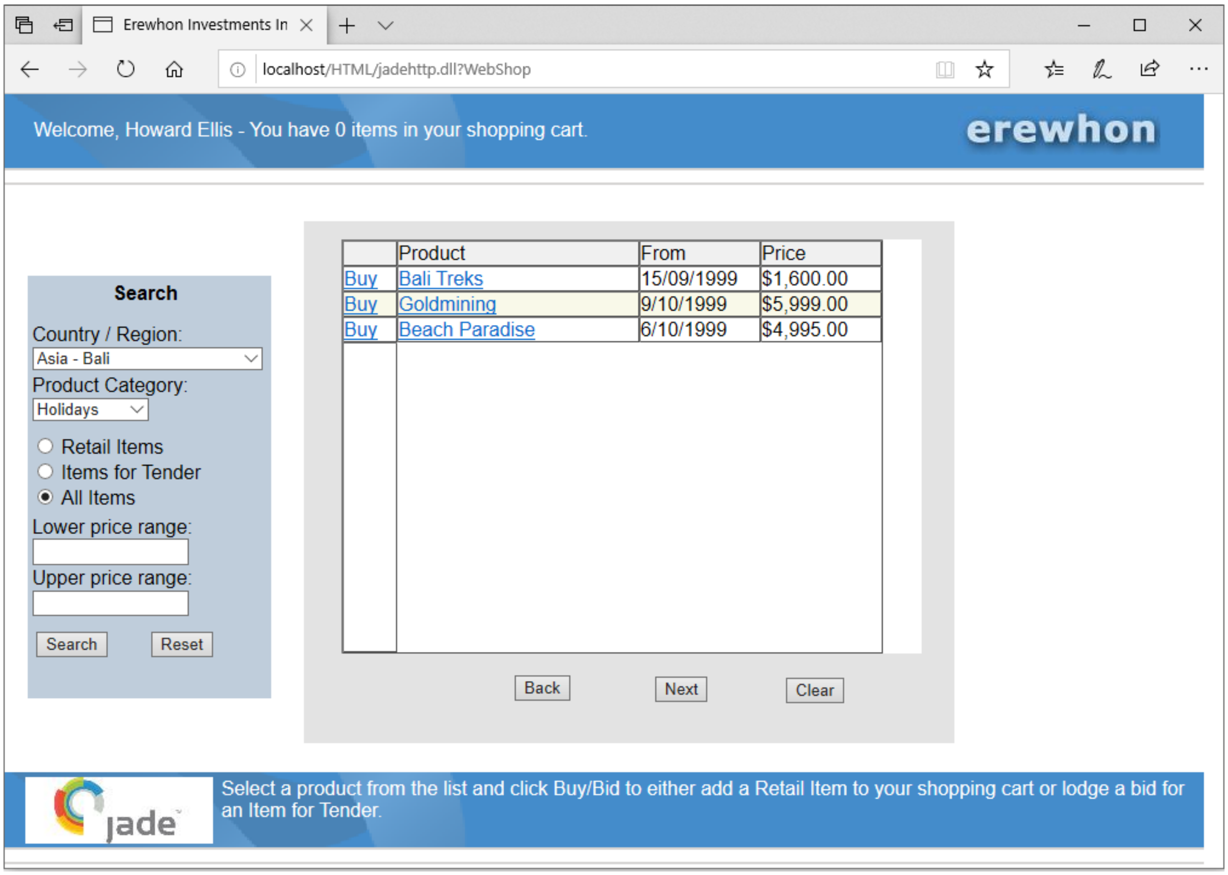
Welcome to the Erewhon Web Shop, please select your name from the list.

Please select your Client name from the listbox, and click 'Enter'.

Select a user name from the drop-down list box and then click **Enter**.

Product Search

The Product Search form is the main form of the **WebShop** application. To begin, search for a list of products by selecting the required search criteria in the **Search** panel and then click the **Search** button.



Welcome, Howard Ellis - You have 0 items in your shopping cart.

Country / Region: Asia - Bali
Product Category: Holidays
Retail Items
Items for Tender
[✓ All Items]
Lower price range:
Upper price range:

	Product	From	Price
Buy	Bali Treks	15/09/1999	$1,600.00
Buy	Goldmining	9/10/1999	$5,999.00
Buy	Beach Paradise	6/10/1999	$4,995.00

Select a product from the list and click Buy/Bid to either add a Retail Item to your shopping cart or lodge a bid for an Item for Tender.

A new search list can be generated at any time by changing the search criteria and then clicking the **Search** button. The search criteria can also be reset by clicking the **Reset** button. The list of search results can be cleared at any time by clicking the **Clear** button below the list of results (this will not affect any of the items currently in the shopping cart). To scroll through the search results list, click the **Next** button or the **Back** button.

Some products are retail sale items, while others are tender items for which you must enter a bid.

Viewing the Details of a Product

The second column in the search results list is the name of the product item, which is a link to the details of the product. By clicking on the product item's name link, the details of the product will be displayed. If the selected product is a tender item, then as the details of the product are displayed, a field will also be shown in which to enter your offer (your tender amount).

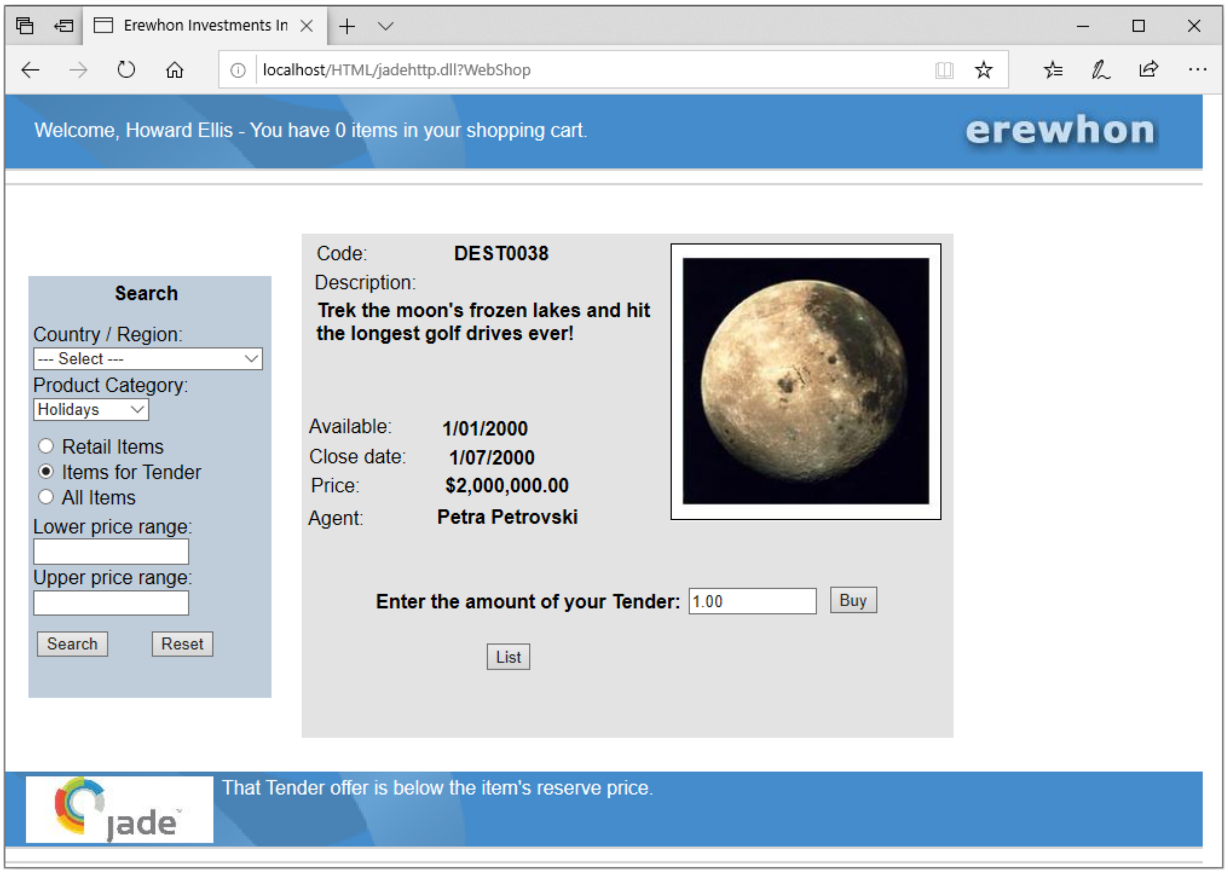
Buying or Bidding for a Product

The first column in the search results list will contain a link named **Bid** or **Buy**. Click this link to purchase a sale item (buy) or bid for a tender item. If you are buying a product item, it will be added to your shopping cart immediately. If the selected product is a tender item, the details of the product will be displayed and a field will also be shown, in which to enter your offer (your tender amount).

After entering your offer, click the **Buy/Bid** button and the tender will then be added to your shopping cart. To return to the search results list, click the **List** button. To go to the checkout, click the **Checkout** button.

Product Details/Tender

When a product is selected in the search results list by clicking the product name, more-detailed information about the product item will be shown. Click the **List** button to revert to the search results list.



Welcome, Howard Ellis - You have 0 items in your shopping cart.

Country / Region: --- Select ---
Product Category: Holidays
Retail Items [✓ Items for Tender] All Items
Lower price range:
Upper price range:

Code: DEST0038
Description: Trek the moon's frozen lakes and hit the longest golf drives ever!
Available: 1/01/2000
Close date: 1/07/2000
Price: $2,000,000.00
Agent: Petra Petrovski

Enter the amount of your Tender: 1.00

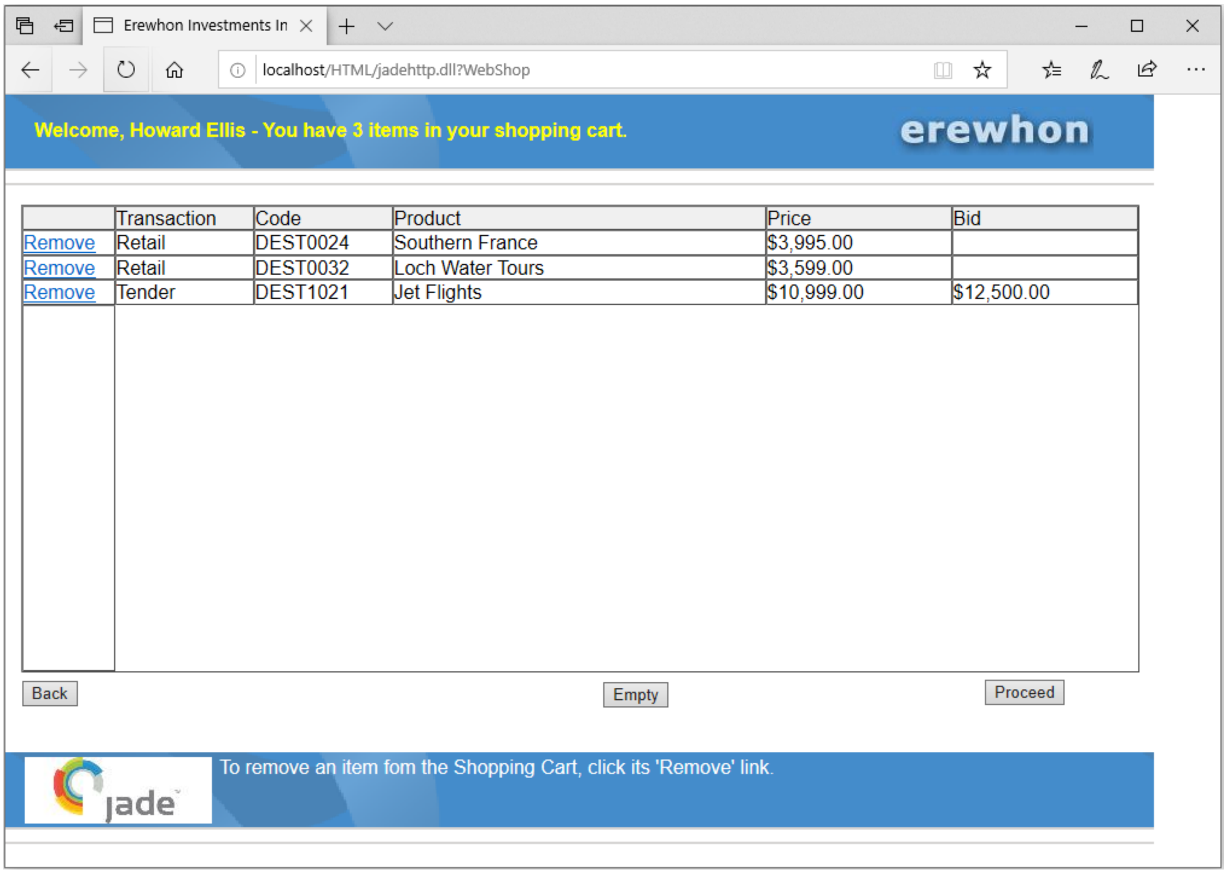
That Tender offer is below the item's reserve price.

If the product item is a tender item, the details of the product will include a field in which to enter your offer (your tender amount).

Clicking the **Reset** button with the product details in view will cause the product details to be replaced with an empty search results list and the search criteria will also default to their original settings. If the **Search** button is clicked with a product's details displayed, the details will be replaced with the new search results list.

Checkout

The checkout is the final confirmation of your shopping cart before proceeding with the transaction of purchasing sale items or submitting any bids for tender items, or both purchasing sale items and submitting any bids for tender items.



Welcome, Howard Ellis - You have 3 items in your shopping cart.

	Transaction	Code	Product	Price	Bid
Remove	Retail	DEST0024	Southern France	$3,995.00	
Remove	Retail	DEST0032	Loch Water Tours	$3,599.00	
Remove	Tender	DEST1021	Jet Flights	$10,999.00	$12,500.00

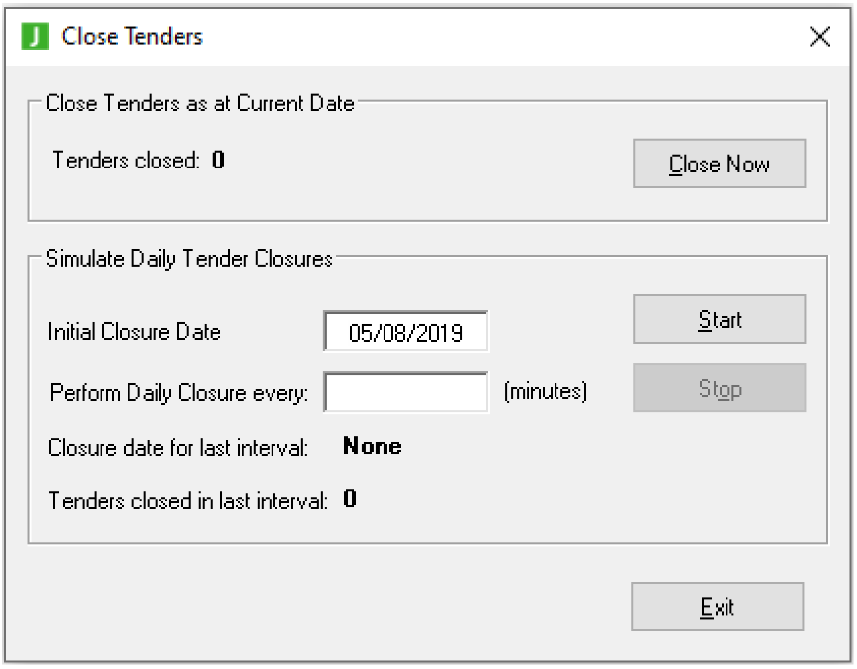
To remove an item fom the Shopping Cart, click its 'Remove' link.

To remove any unwanted items from the shopping cart at this point, click the **Remove** link (underlined) in the first column of the item's row. To remove all of the items in the shopping cart, click the **Empty** button. If you want to return to the Product Search form, click the **Back** button. To initiate the final processing of the shopping cart, click the **Proceed** button. A list of the bought and tendered items will then be displayed, so that you can review the transaction.

Tender Closure Application

The **Tender Closure** application runs the processing that converts the highest tenders for sale items into actual sales. It does this by processing all tender items at a specific date, and if the tender item's closure date is on or prior to the specified date and the item is not sold, the highest tender offer is converted into a sale. Typically, such an operation would be implemented as a batch (separate) process that runs outside the main applications (for example, it can be implemented as a separate application that schedules the processing to be run once a day, late at night), which is why we have implemented this processing in a separate application.

The application has only one form, shown in the following image.



Close Tenders

Close Tenders as at Current Date

Tenders closed: **0**

Simulate Daily Tender Closures

Initial Closure Date 05/08/2019

Perform Daily Closure every: (minutes)

Closure date for last interval: **None**

Tenders closed in last interval: **0**

Enter the date in the **Close tenders as at date** text box at which tenders are to be closed. Any unsold tender items with a closure date on or prior to this date will be processed.

To process tenders immediately, click the **Close Now** button. The operation will start and when it completes, the number of closed tender items will be displayed. The application also gives an example of how to use JADE timers to schedule processing. Enter a number of minutes in the **Closure interval** text box and then click the **Start** button. This will start a timer that counts down from the specified number of minutes, with progress being displayed at the bottom of the form.

When the time period expires, any unsold tender items with a closure date on or prior to the date specified in the **Close tenders as at date** text box will be processed. The number of closed tender items will be displayed. The closure date will then advance one day and the timer will restart from the specified number of minutes. This allows you to simulate the scheduling of the operation to run once a day. To stop the timer, click the **Stop** button (which is enabled only when the timer is active). To shut down the application, click the **Exit** button.

Part 3 Model Implementation

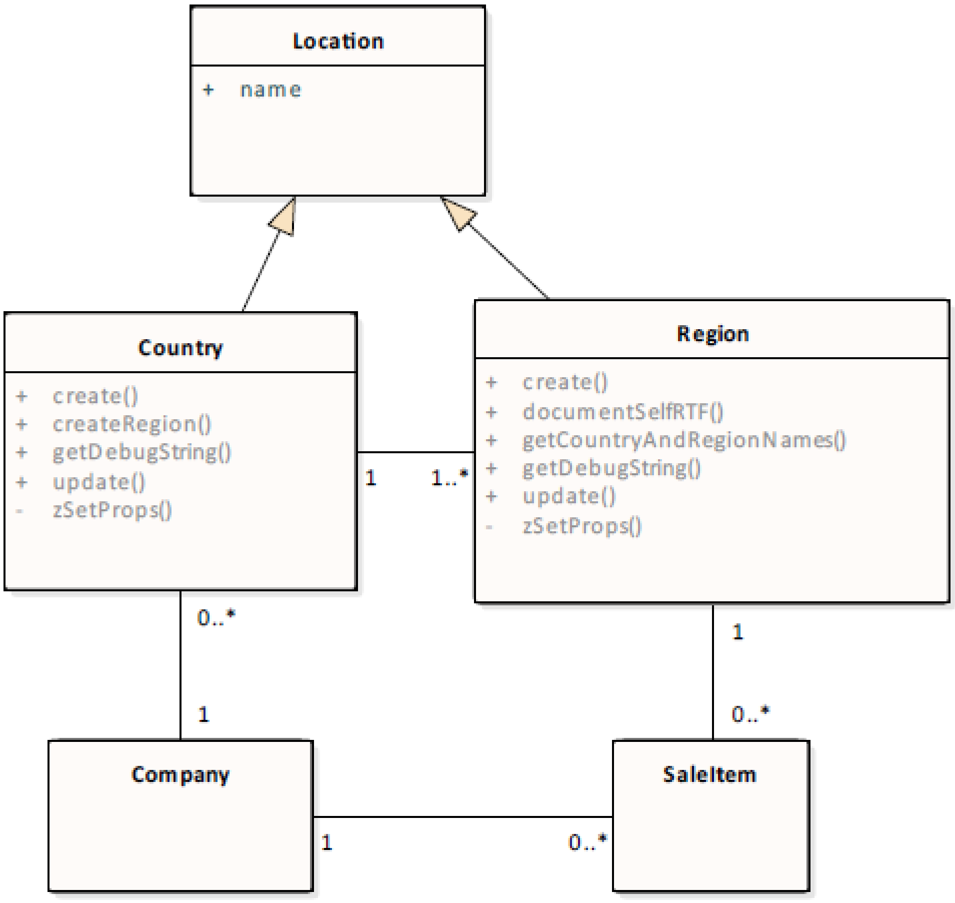
This section describes the model entity classes that implement the core object model of the Erewhon Investments system. All of these classes are defined in **ErewhonInvestmentsModelSchema** and they inherit from a common superclass, **ModelEntity**, as follows.

- ModelEntity (abstract)
    - Address
    - Agent
    - Client
    - Company
    - CommissionRate
    - Location (abstract)
        - Country
        - Region
    - Sale (abstract)
        - RetailSale
        - TenderSale
    - SaleItem (abstract)
        - RetailSaleItem
        - TenderSaleItem
    - SaleItemCategory
    - Tender

Diagrams describe the relationships between these classes, followed by an overview of each concrete class. For more details, see the following subsections.

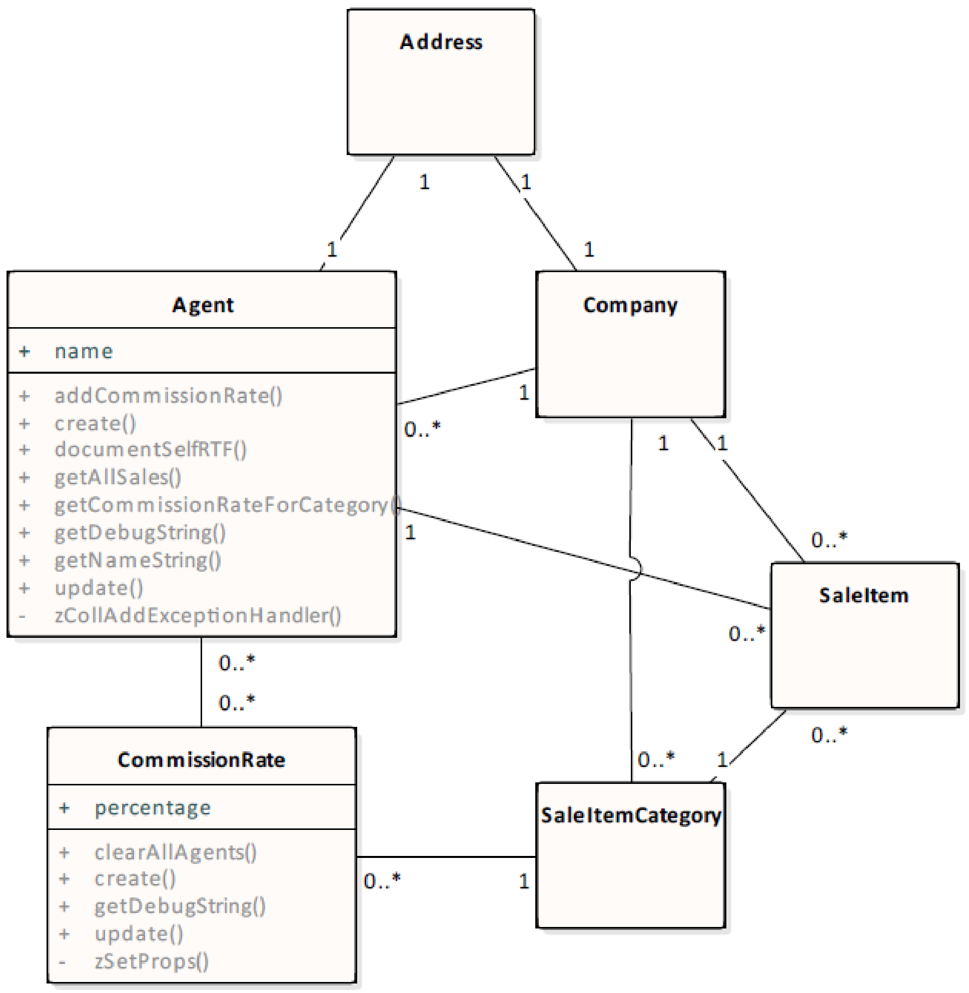
Locations

The following diagram describes the relationships between **Location**, **Country**, and **Region** and their related **ModelEntity** classes.



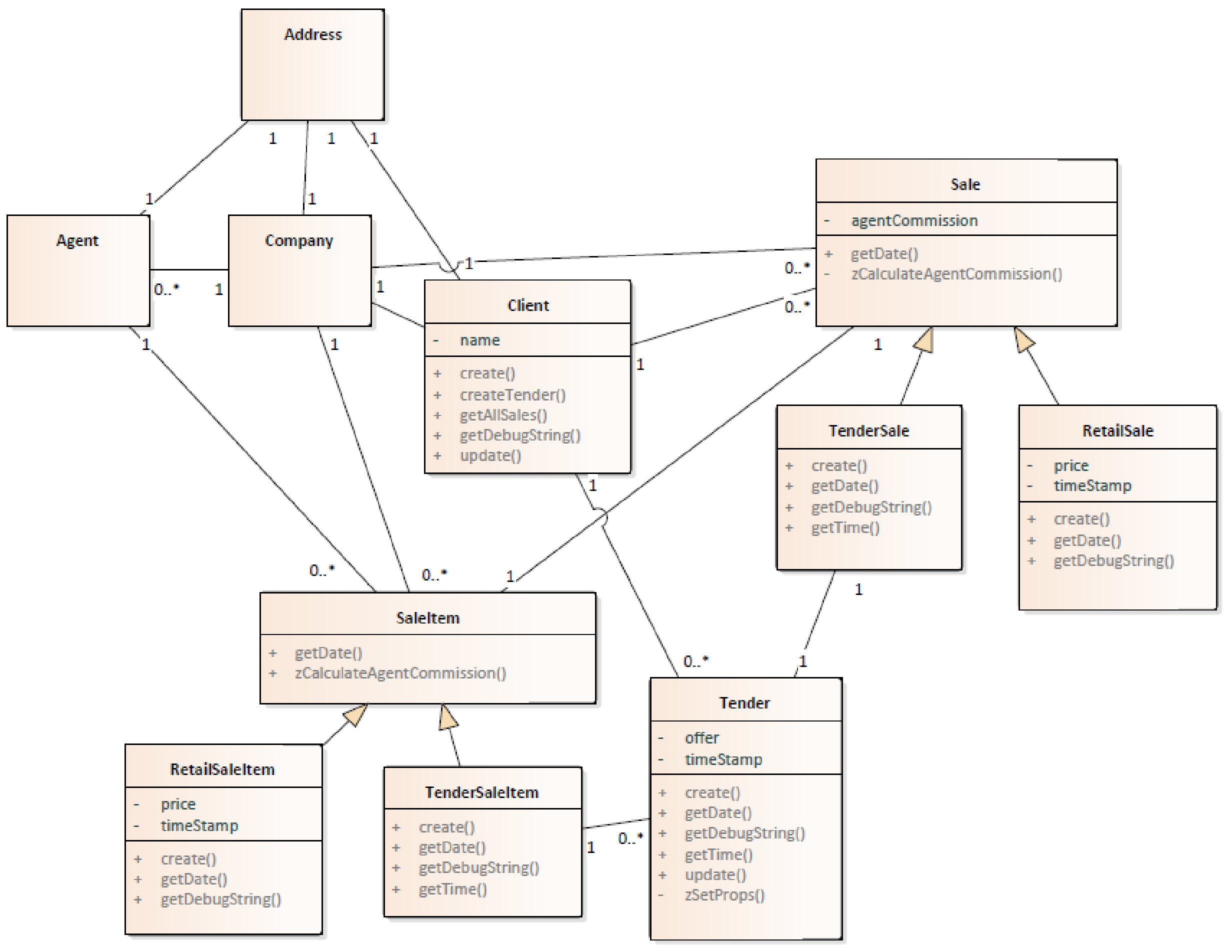
Agents and Commission Rates

The following diagram describes the relationships between **Agent** and **CommissionRate**, and their related **ModelEntity** classes.



Sales and Clients

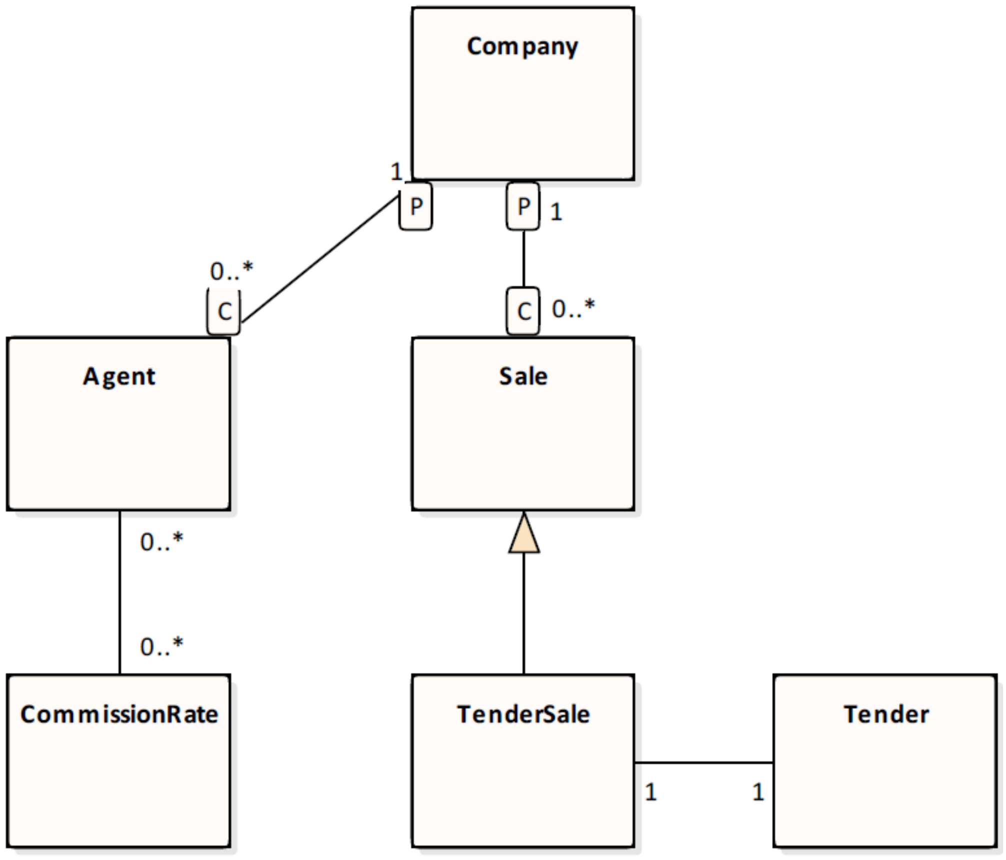
The following diagram describes the relationships between **Sale** (and subclasses), **SaleItem** (and subclasses), **Client** and **Tender**, and their related **ModelEntity** classes.



JADE Reference Diagram

The following diagram describes the JADE implementation of the relationships described above. In JADE, a two-way relationship is implemented by defining an *inverse* between two reference properties. Single-value references are used to implement the “one” side of a relationship; collection references are used to implement the “many” side. In this way, one-to-one, one-to-many, and many-to-many relationships can be implemented.

This diagram has an example of at least one relationship of each cardinality. Cardinalities are represented by the standard UML syntax, except that **P** and **C** represent a parent/child relationship, where **P** is the parent.



Agent

Agents represent users of the system who bring in items for sale. These items can be offered for retail sale or for sale by tender. An agent can have many items for sale at any one time. For each sale item category in the system, an agent operates at one (and only one) commission rate. The commission rate from which an agent is operating for a category determines the percentage of each sale from the category that an agent takes as commission.

Client

Clients represent users of the system who log on to search for and purchase items. Clients can purchase retail items immediately, or place bids on items offered for sale by tender. Once a tender sale item is closed, the highest tender is converted into a sale for the client. Clients know about all sales in which they have been involved, and all tender offers they have made.

Company

A single instance of **Company** provides the root object for the system. We assume only one persistent company instance at any one time and the **create** (constructor) method of **Company** enforces this. **Company** represents the top of the parent-child reference hierarchy, and provides collections through which we can navigate to all other objects in the system.

Commission Rate

One or more commission rates can exist for each sale item category. Each commission rate can have multiple agents operating from it. A commission rate determines the percentage commission its agents make on sales from the commission rate's category.

Country

Countries provide a means of grouping and organizing geographical regions. Each country can have multiple regions defined for it and as such, may or may not represent an actual country; for example, a continent that has relatively few regions may be counted as a country.

Region

Regions provide a means of grouping sale items into geographical areas. Each region is owned by a country and can have multiple sale items located in it.

Retail Sale

Retail sale objects model sales of retail items. Each retail sale has a price and a time stamp, and inherits its client, sale item, and company references from the **Sale** class. A retail sale knows the item that was sold and the client to whom it was sold. The agent's commission on the sale is calculated at the time the sale is created.

Tender Sale

Tender sales represent sales of items offered for sale by tender. A tender sale is created when the closure date on a tender sale item has passed, and the highest tender is accepted and converted into a sale. Each tender sale knows the item sold, the client to whom it was sold, and the winning tender object. The agent's commission on the sale is calculated at the time the sale is created.

Retail Sale Item

Instances of this class represent items offered for retail sale. All items are owned by the company and are organized into categories and geographical regions. Each item knows the agent who brought it in for sale. All sale items have a two-part code consisting of a string prefix followed by an integer number. The prefix is supplied by the application, while the number is allocated automatically when an item is created. Items can also hold a 200 by 200 pixel image of themselves.

Once an item has been sold, its **mySale** property refers to the sale in which it is involved. If **mySale** is not assigned (that is, it has a null value), the item is not yet sold.

Tender Sale Item

Instances of this class represent items offered for sale by tender. All items are owned by the company and are organized into categories and geographical regions. Each item knows the agent who brought it in for sale. All sale items have a two-part code consisting of a string prefix followed by an integer number. The prefix is supplied by the application, while the number is allocated automatically when an item is created. Items can also hold a 200 by 200 pixel image of themselves.

A tender sale item has a minimum (reserve) price, offers below which will not be accepted. The item's closure date indicates the date at which bidding will stop. At this date, the highest tender is accepted and a sale is created for the item. A tender sale item knows all bids that have been made for it. Once an item has been sold, its **mySale** property refers to the sale in which it is involved. If **mySale** is not assigned (that is, it has a null value), the item is not yet sold.

Sale Item Category

Sale item categories allow items to be grouped into logical categories. All categories are owned by the **Company** object.

Categories also hold all of the commission rates at which agents can operate for sale items belonging to the category.

Tender

Tender objects represent bids made by clients on items offered for tender sale. A tender holds the offer price and time stamp of the bid, as well as the client who made the bid and the item for which they have tendered. If a tender is accepted when bidding for an item is closed, a tender sale object is created and **myTenderSale** will be set to this object. If **myTenderSale** is non-null after bidding on the tender's item has closed, it means that the tenderer won the item.

Part 4 Design Considerations

This section discusses some of the design issues considered during implementation of the Erewhon Investments demonstration system. It focuses on those we feel are most important, and as such should not be seen as an exhaustive discussion of all design issues. What we propose in this document are guidelines.

We have tried to illustrate as many points as we can in the Erewhon system, without making it too complex. At the same time, it is intended to be a working multiuser system that deals with a number of issues encountered when building production applications. This is essential to illustrate the rationale of our design decisions.

What we're trying to stress are the design issues themselves, or *themes*. The demonstration system is just one possible implementation. There are undoubtedly many other ways of addressing the problems we shall discuss. We have tried to keep things as straightforward as possible to make it easier for those new to JADE, but without trivializing the points.

The main thing to take away from this document is an awareness of some of the issues that should be considered early on in a development project.

For details about design considerations, see the following subsections.

Conventions

Before we get underway, we should point out some of the conventions used in the Erewhon system. These should be seen as guidelines only, as several are subject to personal preference.

- All protected property and method names start with a lower case **z**. We use this to distinguish them from public features, and to force them to appear at the end of property and method lists.
- Single-value (that is, non-collection) references are prefixed with **my**.
- Multi-value (that is, collection) references are prefixed with **all**.
- Global constants are used extensively for such things as error numbers, application names, and version numbers. From the Browse menu in JADE, select **Global Constants** command to view the global constants for a schema.
- Except for development, testing, or peripheral methods, literal strings are not used in code. Instead, strings are defined as translatable strings. To view translatable strings for a schema in JADE, select the **Strings** command from the Schema menu.
- In general, we prefer to make properties read-only rather than implement *get* methods for them (see "Model Operations", later in this document).

Models, Views, and Controllers

The Model, View, Controller architecture (MVC) was popularized by Smalltalk. It divides a system into an underlying model, any number of different views of the model, and controllers that synchronize interaction between the model and the views. MVC makes it possible to concentrate on the essentials of a system (the model), and add the application and user interface layers independently. There can be many different view and controller pairs for each model; the intention is that views and controllers can be modified extensively with little or no change in the model.

For many systems, though, the role of the controller is small, with little or no distinction between views and controllers in terms of implementation. While they always represent distinct concepts, often they are implemented as one. Views can take responsibility for their own synchronization and sometimes the model provides synchronization services. The Erewhon system is an example of this. In such cases, we can simply refer to the Model and the Views (MV).

Model and View Separation

The model and views have been separated into their own schemas (which are discussed in the following section). This makes explicit the distinction between the model and its views.

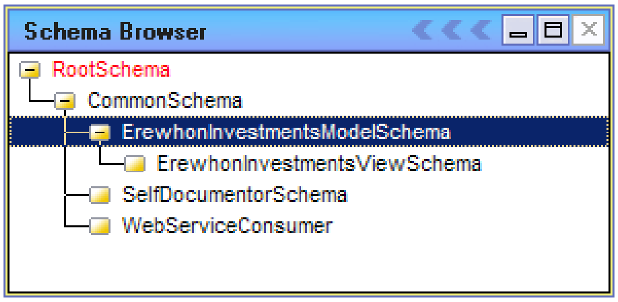
The model should focus on the problem (or business) domain. The question to start with is “How do we best model the business operations?”. By separating the model from the views, the model can be made more independent of application-level and user interface requirements. Separating the model allows you to build a more-stable base, since the business domain (which the model represents) is generally less likely to change than the application layers or the user interface, or both the application layers and the user interface. A well-defined model can support several applications. For example, in the Erewhon system we have one model schema, with the view schema defining four applications that run over this model.

We have used subschemas in JADE to separate the model from the views. It allows for a cleaner, more well-defined design and implementation. It also means that separate development teams can more easily work on separate parts of the system, but still within the same single JADE environment. Separating the views from the model by packaging them in their own schemas prevents the model schema from becoming cluttered with user interface implementation, and means that the model schema can support many different views. It also makes it easier to identify the services provided by the model.

Schemas

A schema is the highest-level organizational structure in JADE and represents the object model for a specific domain. A schema is a logical grouping of classes, together with their associated methods and properties. These effectively define the object model upon which applications are based. The appearance and functionality of applications in a schema can differ, but they all share the underlying object model defined by the schema. JADE provides the **RootSchema**, which is always at the top of the schema hierarchy. The **RootSchema** provides essential system classes that are available to all subschemas.

The schemas that make up the Erewhon Investments system are shown in the following image of the JADE Schema Browser window.



The schema hierarchy is analogous to a class hierarchy and similar terminology is used. For example, **ErewhonInvestmentsModelSchema** is a subschema of **CommonSchema**, and **ErewhonInvestmentsViewSchema** is a subschema of **ErewhonInvestmentsModelSchema**. Subschemas inherit all the classes, methods, and properties that are defined in their superschemas. Therefore, all schemas in the Erewhon system inherit the entities defined in the **CommonSchema**.

The system is implemented in three main schemas and two supplementary schemas, as follows.

CommonSchema

This inherits from **RootSchema** and provides common services for all of its subschemas. Services include common exception-handling methods on the **GCommonSchema** class and a selection of subclassed controls, including a date text box and collection viewer list box.

ErewhonInvestmentsModelSchema

This schema implements the model for the system. All classes for which persistent objects are created are defined in this schema as subclasses of **ModelEntity**. The schema also implements a number of classes that provide services to the views including transaction agents and order proxies. A number of utility **JadeScript** methods are provided in this schema for development and testing use, such as methods to initialize the database.

ErewhonInvestmentsViewSchema

This schema implements the views or applications that run over the model. The entire user interface is implemented in this schema. The schema defines six applications.

- **Administration** is a back-office application that company staff and agents can use to administer the system. It is expected that this application would be deployed on a mixture of standard clients and JADE thin clients.
- **ErewhonShop** is a front-office application that clients will use to search the items for sale, and to buy or bid on items. It is expected that this application would be deployed on JADE thin clients, but the application can obviously be run on standard clients as well.
- **WebShop** is a Web application server that allows clients to search, buy, and bid on items from within a Web browser. It provides similar functionality to the **ErewhonShop** application but with a slightly different interface for Web browsers.
- The **TenderClosureApp** closes all sale items open for tender if their closure date is at or prior to a specified date. The application closes each tender sale item and accepts the highest bid.

  You can specify a timer interval so that the operation is performed automatically on a regular basis. We expect that one copy of this application would run on a standard client (possibly on the same machine as the server) and would be set to run the operation at a time when activity is low.
- **WebServiceOverHttpApp** and **WebServiceOverTcpApp** are applications that demonstrate JADE's Web service provider capabilities. For details, see the *SOAP Web Services* white paper (which is also available from the JADE Web site at https://www.jadeworld.com/developer-center/resource-library/white-papers).

The view schema also extends model classes by adding methods to them in the view schema. For example, **display** and **getSearchResultString** methods are added to **ModelEntity** subclasses in the view schema.

SelfDocumentorSchema

This schema demonstrates JADE interfaces and JADE packages by exporting a framework that allows objects to document themselves. Refer to the **FormDocumentorSetup** form in the **ErewhonInvestmentsViewSchema**, to see how this package and the interface that it exports is used.

The **btnShow_click** method on the form invokes the package. This form can be accessed by running the **Administration** application from the view schema and selecting the **Misc | Show Details via Interface** menu item.

WebServiceConsumer

This schema demonstrates JADE's Web service consumer capabilities. For details, see the *SOAP Web Services* white paper (which is also available from the JADE Web site at https://www.jadeworld.com/developer-center/resource-library/white-papers).

Transaction Separation

The **TransactionAgent** class in the model schema provides activity methods that implement all transactions in the model. Except for development or peripheral tasks (for example, initializing the database), there are no other places in the model that begin or commit transactions. Our views also should rarely (if ever) begin and commit their own transactions. Rather, transaction methods should be added to the **TransactionAgent** in the model, or to the **TransactionAgent** subschema copy class in the view schemas.

Transactions are a concept in their own right. A transaction brackets one or more model operations into one activity (logical unit of work) bounded by begin transaction, and commit or abort transaction. Each operation in the model should generally not be responsible for going into and out of transaction state, as this is error-prone and reduces the flexibility by which operations can be combined into transactions or activities.

It is important to recognize that transactions are application-defined. The application requirements determine the boundaries of a transaction, as it is the application that determines what is a logical unit of work. In the Erewhon system, we have defined the **TransactionAgent** class in the model schema because we have only one set of transactions for the views. We thought it made the system an easier example to understand if the transaction methods were defined in the model schema along with the operations that they call. However, it is important to note that although they are located in the same schema, the transaction methods are distinct from the model. They are determined by the requirements of the applications in the view schema.

Indeed, for a larger system, it may be appropriate to implement transactions in their own subschema between the model and the views. Alternatively, each view can implement its own **TransactionAgent** class (or an equivalent mechanism). For systems with many transactions or complex transactions, creating several specialized **TransactionAgent** (or equivalent) subclasses is an approach worth considering. Some systems even go as far as defining classes to represent individual transactions. The technique that you adopt depends on the specific requirements of your application.

Separating out transactions in this way brings several benefits, as follows.

- Transaction code is centralized.
- Model operations do not have to worry about beginning, committing, or aborting transactions.
- Model operations can be more-easily combined into different transactions or activities.
- It provides a centralized layer for enforcing certain lock policies.
- It provides a layer to encapsulate exception handling.

Model Operations

Operations in the model should be based on business application requirements. In **ErewhonInvestmentsModelSchema**, operations are implemented as methods on the **ModelEntity** subclasses.

We have avoided *unconditionally* defining *get* and *set* methods for all **ModelEntity** subclass properties. It is not uncommon for projects (almost religiously) to insist on defining all properties as protected and implementing public *get* and *set* methods. However, this tends to overlook certain higher-level JADE concepts that provide an effective alternative without sacrificing encapsulation or flexibility, and with lower runtime overhead. In general, we have defined properties that are part of the public interface of a class as read-only, for the following reasons.

- A property, when accessed in the JADE language, is already equivalent both conceptually and in reality to a pair of related *get* and *set* operations implemented by the JADE Object Manager. When you refer to a property using the JADE language, you are doing so via the default *get* and *set* operations provided by the Object Manager; never directly.
- The access option for a property defines which of the implicit *get* and *set* operations are part of the public interface of the class; protected implies none, read-only implies get only, and public implies both.
- If you insist on defining and writing *get* method wrappers for *all* properties that simply return the property value, this does not really increase encapsulation. All it does is incur the unnecessary runtime overhead of dynamic binding and method dispatch.

It is accepted, though, that for implementation encapsulation (that is, information hiding) *get* methods are appropriate. However, these should be identified on a case-by-case basis. For example, the **Sale** class in **ErewhonInvestmentsModelSchema** implements a **getAgentCommission** method that simply returns the value of the **zAgentCommission** property. The implementation of sale commissions has in fact changed a couple of times from being a value derived when the *get* is requested, to being a value calculated and stored on the sale when it is created. These changes indicate that in this case, having a *get* wrapper for the agent commission is warranted but implementing a **getName** method on **Address**, for example, instead of simply defining **name** as read-only would seem heavy-handed.

- If you ever need to redefine the behavior of the implicit *get* or *set* operations (without changing the type of the property), JADE has the solution: mapping methods. These can be added at any time, so there is no loss of flexibility.
- The read-only option still imposes the discipline whereby only the methods defined in a class (the implementation) can change the state of its instances (the desirable level of encapsulation).

Unconditionally defining *set* operations for *all* properties defeats encapsulation, as it exposes every property to updates from any other class. This can also give rise to update order dependencies, as the order in which properties are set cannot be controlled if they can be set from anywhere. It is common to set references individually and to set groups of attributes (that is, properties other than references) in a single call. This is what we have implemented in the Erewhon system (see the update methods on the **ModelEntity** subclasses).

Bear in mind that our decision to implement a single update method for attributes on some classes was based on our business/application requirements. Those classes implementing a single update method for attributes represent low-volatility data that will be updated one object at a time from the views. However, imagine, for example, that we have to support a frequent (albeit fictitious) transaction that requires updating just the e-mail address on *all* **Address** objects. It would make sense in this case to have an individual **setEmail** method rather than passing all attribute values to update, knowing that only the e-mail address is going to change. The key point is to determine what *set* or *update* methods you need, and how many properties each of them update, based on the transactions and operations your model must support.

Be wary of defining *set* or update methods for each class that set *all* of its properties (attributes *and* references, as opposed to just attributes) in one call. An exception to this is when an object is first created. For object creation, it is often useful to have a method that sets all properties in one call. The model schema does this by implementing a **create** method on **ModelEntity** subclasses. This method is to be invoked only when an entity is first created. For updates to existing objects, setting all attributes (that is, properties other than references) in one method is common, but setting references should be considered more carefully. The Erewhon model schema implements an update method on several **ModelEntity** subclasses that sets all *attributes* only.

The decision as to what references can be updated (after an object has been created) should be based on business/application requirements. Methods should be defined that represent the operations to be performed, rather than just implementing methods that set all references at once. Having such generic methods makes model operations less clear, reduces encapsulation, and can introduce update order dependencies (increasing the chance of deadlocks). Typically, changing a single reference is a single operation. Of course, that does not exclude the possibility of operations that need to change multiple references. However, methods that do this should be the exception, not the rule.

ModelEntity classes implement specific operations to change those references to which updates are permitted; for example:

- **Agent::addCommissionRate**
- **CommissionRate::clearAllAgents**
- **SaleItem::updateCategory**
- **SaleItem::updateRegion**

Exception Handling

Exception handlers are an effective means of encapsulating code for handling unexpected or infrequent errors. Within a transaction, exception handlers are often responsible for restoring things to a consistent state if something goes wrong (usually by aborting the transaction). They are useful for efficiently guarding against invalid object references on an exception basis, rather than always checking the validity of an object in-line (which can require more round trips to the server and defeat caching).

For a discussion of exception handling in JADE, see the *JADE Exception Handling* white paper (which is also available from the JADE Web site at https://www.jadeworld.com/developer-center/resource-library/white-papers).

Exception handling is used extensively in the Erewhon system. Some examples are:

- **TransactionAgent** methods arm handlers to catch exceptions and translate them into error numbers that are returned to the views. This shields the views from having to implement their own exception handling around transaction requests. For examples of this, look at any **TransactionAgent** method, the **TransactionAgent zExceptionHandler**, **zLockExceptionHandler**, **zSilentLockExceptionHandler** methods, and the **ActivityAgent zRegisterObjectAndErrorCode** method.
- The model schema **Application** subclass **ErewhonInvestementsModelApp::initialize** method arms a generic global exception handler and a generic global lock exception handler that are used to catch any exceptions not caught locally. The **GCommonSchema** class in **CommonSchema** provides both of these exception handler methods. The **commonExceptionHandler** method gives an example of a simple generic exception handler and the **commonLockExceptionHandler** gives an example of a lock exception handler. The view schema **GErewhonInvestmentsViewSchema** class reimplements the **commonExceptionHandler** method to perform some exception handling specifically for the **WebShop** application.
- The **ModelEntity** class implements **zCollAddExceptionHandler** to safely add an object to a collection when it is already there and **zCollRemoveExceptionHandler** to safely remove an object from a collection when it is not there. To see uses of these methods, select them in the Class Browser window in JADE and then select **References** from the **Methods** menu.
- The **FormClientApp** class in the view schema implements a **zInvalidObjectExHandler** method that catches all invalid object or deleted object exceptions and redisplays the current form. This exception handler is armed at the start of **FormClientApp** subclass event methods.

Cache Synchronization

When an application references a persistent object, JADE first looks to see if the object is resident in local cache. If it is, the cached object is used for the current operation. If the object is not in cache, it is fetched from the server, brought into cache, and used for the current operation. Once an object has been brought into cache, it is available for use in subsequent operations. Objects do not exist in cache indefinitely. JADE can discard objects from cache when required, to make space for objects being brought into cache. JADE also provides facilities for you to manually discard objects. When an object is discarded, the next reference to it will cause it to be fetched again from the server. In a multiuser system, a locally cached object can be made obsolete when another user updates it. A caching strategy is necessary to keep locally cached objects synchronized with the database, when required.

The Erewhon Investments applications are multiuser and therefore require a caching strategy. A good caching strategy ensures that the objects stored in local cache are the latest editions where necessary, and that this is maintained with the minimum amount of network and processing activity.

An application that makes efficient use of cache will have significant performance advantages over one that does not, as JADE's use of cache is one of its key strengths. A caching strategy comprises all of the mechanisms you use to synchronize local cache with the JADE database.

In the Erewhon Investments system, we have the following considerations.

- While JADE has facilities for developers to manually request that objects be resynchronized in local cache, the automatic cache coherency provided by JADE makes life much easier for developers, and Erewhon takes advantage of this feature. Readers familiar with earlier versions of Erewhon will notice how much code the automatic cache coherency eliminates!

  Automatic cache coherency is enabled by adding the following lines to the JADE initialization file (an example **jade.ini** file for the Erewhon system is provided in **examples/erewhon/erewhonjade.ini**).

  ```
  [JadeServer]
  AutomaticCacheCoherencyDefault=true
  AutomaticCacheCoherency=ServerDefault

  [JadeClient]
  AutomaticCacheCoherency=ServerDefault
  ```

  With automatic cache coherency enabled, objects updated in other nodes (database server, application servers, background nodes, or standard clients) are automatically reloaded in local cache.

- Each operation in the model (that is, methods defined on **ModelEntity** classes) must handle its own integrity locking. By integrity locking, we mean that each operation is responsible for locking those objects of which it requires the latest editions in order to ensure data integrity. Each method assumes responsibility for its own integrity so that the operation is safe, regardless of the context in which it is invoked. Any synchronization locking (that is, locking specifically to serialize transactions) will be done in the respective transaction methods; for example:

  ```
  TransactionAgent::trxCloseTendersAtDate
  ```

- Any **TransactionAgent** method that allows an object to be updated must provide a mechanism for the caller to request that it performs an edition check. This allows the **TransactionAgent** method to verify, on behalf of the caller, that the expected edition of the object is being updated.

- Outside of **TransactionAgent** methods, we are concerned primarily with ensuring that application forms are kept synchronized when objects that they are viewing change and that we have the latest edition of an object before comparing it against search criteria.

We have made use of several mechanisms to implement our caching strategy, in order to illustrate some of the approaches available to you in JADE:

- Automatic cache coherency
- **listCollection**
- **CollectionListBox** subclassed control
- Object notifications
- Edition checking

These are discussed in the following sections.

listCollection

The **listCollection** method of the **ListBox** and **ComboBox** classes (provided by the **RootSchema**) enables list box or combo box controls to have a collection attached to them. Logic attaches the collection to the list box or combo box by using the **listCollection** method. If you use this method to attach a collection to a list box or combo box, little is required to load entries into the list.

If the list box is not sorted, an entry is retrieved from the collection only when it is to be displayed or accessed by logic. Only a few entries from the collection are therefore initially accessed, instead of the entire contents of the collection (though if the list box is sorted, every element in the collection must be accessed). However, as you scroll through the collection, list box entries are not discarded, which means that for large collections, the list box can contain an unacceptably large number of entries. For this reason, **listCollection** should be used only for small collections that will never contain too many items.

When you call **listCollection**, you specify **true** or **false** for an **update** parameter. If the **update** parameter in the **listCollection** method is **true**:

- Deleting the collection results in the list box or combo box being cleared and the collection is no longer associated with the list box or combo box.
- Any changes to the collection cause the contents of the list box or combo box to be discarded and the collection is rebuilt to the current display point (the current entry is reselected if it still exists).

If the **update** parameter is set to **false**, the list box or combo box is not updated and can contain out-of-date information.

The view schema makes use of **listCollection** in several of its forms (for example, the **zInitialize** method of **FormCommissionRate** and **FormLocationsList**). By setting the **update** parameter to **true**, the individual controls handle synchronization of the data they are displaying.

Note We assume that we will never have a large number of commission rates and locations. If this were not the case, use of the default **listCollection** would not be appropriate.

CollectionListBox Class

The **CollectionListBox** class in the **CommonSchema** presents an example of a subclassed control. It implements a **ListBox** subclass that can view a collection in subsets of its members. It implements the **listCollection** method (described earlier in this document) so that it presents the same interface as standard JADE list boxes and combo boxes. Once a collection has been registered with the **CollectionListBox** (using **listCollection**), it takes care of loading elements from the collection as required, depending on the scroll position (the entire collection is not loaded). As you scroll through a collection, members of the list box that are no longer visible are discarded. In this way, the **CollectionListBox** can view collections containing thousands of items without the actual list box contents ever growing to be too large. The **CollectionListBox** will register notifications on the collection and the members displayed in the list, so that it can automatically synchronize itself if they change.

While the **CollectionListBox** is capable of viewing very large collections, if the collection is very big, the time taken for the list box to position itself in the collection when scrolling can become quite noticeable. However, this occurs only with collections of thousands of elements and you would have to question the appropriateness of displaying that many entries in a list box in the first place.

Forms in the view schema make good use of the **CollectionListBox** to display information (for example, **FormAgentClientList** and **FormSaleItemCategoryList**). By using **CollectionListBox**, the forms do not need to worry about synchronizing this information. They can let the list box do it.

Object Notifications

If there are individual objects for which you want to implement specific behavior when they change (such as updating a view), you can use object notifications to manage that part of your caching strategy.

The **CollectionListBox** control class (described earlier in this document) in the **CommonSchema** uses object notifications to be informed of updates to objects that it is displaying, so that it can update itself if they change.

The **CollectionListBox** class begins object notifications in its **zLoadSubset**, **zLoadSubsetReversed**, and **zSetCollection** methods. An example is:

```
if showUpdates then
    // We want to be told about changes to this object
    beginNotification(obj, Object_Update_Event, Response_Continuous,
                      NotifyInstanceUpdate);
endif;
```

If the **obj** object is changed, the list box will receive a notification upon which it can update itself. JADE calls the **sysNotification** method when the notification is received.

Edition Checking

There are several forms in the **Administration** application that present an object to the user, enabling them to edit it. In a multiuser application, we must guarantee integrity by preventing two users from editing the same object at the same time, or by preventing the changes of one user being overwritten by the changes of another (who may have made his or her changes based on an obsolete object). There are several approaches, as follows.

- Share lock the object being edited as soon as the form is displayed. When the user goes to commit his or her changes, try to get an exclusive lock. If the exclusive lock cannot be obtained, display an error. This approach presents the problem that the object might be locked for a long period (for example, if the user takes a long time to make his or her changes, or goes out to lunch with the form open). It could also deadlock if two users, who both hold a share lock for an object, try to commit their changes at the same time (as neither user will be able to upgrade his or her share lock to an exclusive lock in order to update the object).
- Reserve lock the object during form initialization. This allows other users read access, but only we can

upgrade the lock to an exclusive lock (which means that only we can update the object). Until we do so, other processes can still read the object. As with the share lock, this approach means that the object might be locked for a long period.

- Share lock and unlock, or resynchronize, the object during form initialization, and register a notification on it. If a notification is received while the user is editing, we display a message saying the object has been changed and discard the user's updates. The uses must start editing again. This approach introduces a timing hole in that a user may be able to commit his or her changes before the notification of an update arrives at the client from the server.
- Use edition checking (described in the following list). We have used this approach in the Erewhon system.

When presenting the user with a form to edit an object, the view does not keep a lock on the object in order to prevent it from being locked for a long period (potentially impacting concurrency). The form resynchronizes the object it is editing when it initializes using the **resynchObject** method (see **FormBase::zResynchObjectAndGetEdition** in **ErewhonInvestmentsViewSchema**).

We cannot allow an update to proceed if the object on which the user based his or her update is no longer current. We use edition checking to implement this, as follows.

1. The **zResynchObjectAndGetEdition** method synchronizes the object and saves its edition on the form.
2. When the form calls the required **TransactionAgent** method to perform the update, it passes in the saved edition (for example, see **FormAgent::zDoAction** and **FormClient::zDoAction**).
3. Each **TransactionAgent** method that receives a non-zero edition parameter first obtains an exclusive lock on the object to be updated. This brings the latest edition of the object into cache and locks it, thus preventing other users from updating it. We exclusively lock the object because we know we are about to update it. If the supplied edition is not equal to the latest edition of the object, we know that another user has changed it and we return **ObjectOutOfDate** to the caller. For examples of this, see the **TransactionAgent** methods **trxUpdateAgent** and **trxUpdateClient** in the model schema.

This approach gives us a good balance between ensuring that we do not process an out-of-date object, without requiring that the object be locked for the whole time the user is in the edit dialog.

Synchronization of Shop Views

The two shop views (all subclasses of **FormClientApp** in the view schema) implement searching and shopping cart facilities. Both of these features hold references to persistent **ModelEntity** objects during the session. As the shop view can be deployed on the Web, we do not want to rely on notifications to synchronize the view. The shop views deal mainly with sale items, clients, categories, and locations. We expect such objects to be deleted only rarely, so have adopted a fairly straightforward approach of using exception handlers to trap object-not-found and object-deleted exceptions.

Each event method on **FormClientApp** and its subclasses arms a local exception handler at the start of the method. It then calls a non-event method to do the processing. If an invalid object is encountered, the exception simply resets the form, gives the user a message, and then resumes. For an example of this, see the **FormClientSaleItems::btnResultsDetails_click** and **FormClientApp::zInvalidObjectExHandler** methods.

Any exceptions not caught locally will be caught by the **commonExceptionHandler** method implemented in the **GCommonSchema** class. This exception handler method is armed globally when an application starts. The view schema reimplements this method in its **GErewhonInvestmentsViewSchema** class. For non-Web applications, this reimplementation inherits the default behavior, which logs the exception and displays an error message box. For Web applications, it simply aborts the current transaction and redisplays the last page.

Locking

Transactions protect against inconsistencies that can occur if something goes wrong within the transaction itself, but they can do so only within a single thread of execution. Whenever two or more database transactions are operating at the same time, there is the risk that they may interfere with each other by modifying the same objects.

Concurrency control is necessary, and for this we use locks.

To protect against inconsistencies, JADE provides mechanisms to lock objects. In JADE, a lock does two things. Firstly, it controls concurrent access to an object. Secondly, locking an object ensures that the latest edition of the object is brought into local cache in the node. In the JADE language, you can use the **exclusiveLock**, **sharedLock**, and **reserveLock** methods of the **Object** class to lock objects. The valid concurrent lock combinations are displayed in the following table.

	Exclusive	**Shared**	**Reserve**
Exclusive	No	No	No
Shared	No	Yes	Yes
Reserve	No	Yes	No

Exclusive locks are also known as *write* locks and shared locks are also known as *read* locks.

Locks can have two durations: session and transaction. Session locks are held until the end of the session (process/application) that acquired the lock or until the lock is explicitly released using the **unlock** method. Transaction duration locks are held until the end (either commit or abort) of the next transaction (at which point *all* transaction duration locks for the process are released) or until the lock is explicitly released using the **unlock** method when not in transaction state (manual unlocks of transaction duration locks within a transaction are ignored).

Ignoring explicit unlocks of transaction duration locks when in transaction state and releasing all transaction duration locks at the end of a transaction is known as two-phase locking. By doing so, JADE avoids the classic “assumed update” problem, by not allowing a second process to update objects modified by a first process until the first process has committed or aborted the entire transaction.

For examples of locking in the Erewhon Investments system, see the methods on the **TransactionAgent** class and **ModelEntity** subclasses in the model schema.

Exclusive Locks

Before an object can be updated, JADE insists that it be exclusively locked. This prevents two processes from updating the same object at the same time. An exclusive lock can be obtained only if there are no other locks in place for the object. When you lock an object using an exclusive lock, no other process can lock (and hence update) the same object. JADE automatically applies an exclusive lock when an object is updated. By default, updated objects are locked automatically for the duration of the transaction.

Shared Locks

A shared lock allows several processes to simultaneously read an object but not update it. Shared locks enable greater concurrency while ensuring that a process never works with obsolete data. If you lock an object using a shared lock, other processes attempting to update the object or explicitly acquire an exclusive lock wait until the lock is released, but can acquire a shared lock or a reserve lock.

Reserve Locks

A reserve lock is available for situations where you intend to update an object but you need to minimize the length of time the object is locked with an exclusive lock. When you place a reserve lock on an object, other processes attempting to acquire an exclusive lock or reserve lock on that same object wait until the reserve lock is relinquished, but those attempting to acquire a shared lock succeed.

Unlocking Objects

You can unlock objects manually. Use the **unlock** method to explicitly unlock an object. Requests to unlock transaction duration locks when in transaction state are ignored. All transaction duration locks are held until the next commit or abort transaction instruction, at which time they are all released, regardless of whether or not they were explicitly released with an **unlock**.

Inverses and Referential Integrity

A *reference* is a property that contains a reference to another object; that is, it is an end-point in a one- or two-directional relationship. The two types of reference in JADE are:

- An *implicit* reference, in which an object references another object and either of the following is true.
  - The referenced object does not contain a reference back to the first object.
  - The referenced object contains a reference to the first object, but the two properties have not been defined as end-points in a two-way relationship.
- An *inverse* (or *explicit*) reference, in which two objects reference each other and the two properties have been defined as end-points in a two-way relationship.

Inverse (or explicit) references are used in JADE to implement relationships between objects. They offer significant advantages in that JADE will automatically handle updating one side of a relationship (an inverse reference) whenever the other side changes. In addition, if one or both ends of a relationship is a dictionary, related elements in the dictionary are automatically updated whenever their keys change. This helps to ensure referential integrity in your model. In fact, in a persistent model, inverse (or explicit) references should be the rule. There should be few cases where they are not used, and in such cases, a good reason for not using them.

In JADE, a reference can refer to a single object, or to multiple objects (via a collection). This allows you to implement one-to-one, one-to-many, and many-to-many relationships. Relationships can be defined as peer-to-peer or parent-child. They differ only when objects are deleted. A parent-child relationship allows you to implement a cascading delete where all related children of a parent object are deleted when the object itself is deleted. In a peer-to-peer relationship, when one object is deleted, all references to it in its related objects are removed (set to null).

JADE allows a reference to have multiple inverses (that is, participate in multiple relationships). In such cases, JADE will automatically propagate updates on a single reference to multiple inverse references.

The **ErewhonInvestmentsModelSchema** employs inverse references extensively. Some examples are as follows.

- One-to-One Relationships
- One-to-Many Relationships
- Many-to-Many Relationship
- Parent-Child Relationships

- Multiple Inverse Relationships
- Automatic Key Maintenance

One-to-One Relationships

SaleItem::mySale to **Sale::mySaleItem**

TenderSale::myTender to **Tender::myTenderSale**

One-to-Many Relationships

Company::allAgents to **Agent::myCompany**

SaleItemCategory::allCommissionRates to **CommissionRate::mySaleItemCategory**

Many-to-Many Relationship

Agent::allCommissionRates to **CommissionRate::allAgents**

Parent-Child Relationships

Company::allClients to **Client::myCompany** (a one-to-many relationship)

Country::allRegions to **Region::myCountry** (a one-to-many relationship)

Parent-child relationships are what allow the **JadeScript** method **deleteAllData** to purge the database by simply deleting the **Company**. JADE cascades the delete through all of the parent-child relationships in the model.

Multiple Inverse Relationships

Tender::myTenderSaleItem to **TenderSaleItem::allTendersByOfferTime**

Tender::myTenderSaleItem to **TenderSaleItem::allTendersByTimeOffer**

Whenever **myTenderSaleItem** is set on a **Tender**, the **Tender** is added to both the **allTendersByOfferTime** and **allTendersByTimeOffer** dictionaries on the sale item.

Sale::myClient to **Client::allTenderSales**

Sale::myClient to **Client::allRetailSales**

These illustrate a conditional multiple inverse relationship. Whenever **myClient** is set on a **Sale**, the **Sale** is added to **allTenderSales** on the **Client** if it is a tender sale (because membership of the **allTenderSales** dictionary is **TenderSale**) and **allRetailSales**, if it is a retail sale (because membership of the **allRetailSales** dictionary is a **RetailSale**).

Automatic Key Maintenance

Client::myCompany to **Company::allClients**

In the above one-to-many relationship, if the name of a client is changed, the **allClients** dictionary of the **Company** to which the client is related will automatically be updated.

SaleItem::myCompany to **Company::allSaleItems**

In the above one-to-many relationship, if the code prefix or code number of the sale item changes, the **allSaleItems** dictionary of the sale item’s company will automatically be updated.

Key Paths

A key path is a mechanism that enables you to define a dictionary key that is not an embedded property of the members of the dictionary, but is instead derived from the member objects. When you define a key path, you specify a chain of references starting from the member class and finishing at an end-point. At run time, the references are traversed to arrive at the end-point that yields the key value. Like all dictionary keys, if the dictionary participates in a relationship, changes to key path keys will automatically be propagated to the related dictionaries.

The **ErewhonInvestmentsModelSchema** has several dictionaries that make use of keys paths. Three of them are:

- **SaleByItemDict**
- **RetailSaleByTimeItemDict**
- **TenderSaleByTimeItemDict**

Server Methods

The **serverExecution** method option indicates that the method and all methods subsequently called by this method are to be executed at the database server node (unless they are **clientExecution** methods, in which case they are executed at the node of the client calling the method).

By simply adding **serverExecution** to a method signature, JADE will shift execution of the method to the database server node. This method option provides performance benefits (by reducing network traffic) when a method accesses a large number of persistent objects in multiuser mode. The methods are executed at the node in which the objects reside, rather than the required objects having to be passed across the network to the client node for processing.

See the **TransactionAgent::trxCloseTendersAtDate** and **Company::closeTendersAtDate** methods in the model schema for an example of **serverExecution**. These methods are used to close all open tender sale items at a specified date, and as such, we expect that they may reference a large number of objects. As this is a *batch*-type operation and there is no requirement for them to be processed at the client node, we implement this transaction as a server method to avoid all of the tender sale items and their associated tenders and related objects having to be brought across the network.

Server methods are great for distributing code to reduce network traffic. However, be aware of the following restrictions.

- Transactions must be total client transactions or total server transactions; that is, any begin and commit transaction pair of instructions must be done while executing on the client (without executing an updating server method), or while executing on the server (without the execution of updating client methods).
- Persistent transactions must be started, performed, and finalized at a single node. All of the update operations of the transaction must occur in the same node that started the transaction.
- Server methods cannot invoke GUI methods.

In the **ErewhonInvestmentsModelSchema**, it is for the first two reasons that the **InitialDataLoader loadData** method commits the first transaction before invoking the **zCloseTendersAtCurrentDate** method (which begins and commits a transaction on the server).

Tender closures are performed in a server method, so they require a separate transaction on the server. The first transaction, which we begin on the client, must be committed on the client before we start the server transaction.

Skins

A *skin* is a series of images that is applied to the caption line, menu line, and border areas of each form to provide an enhanced look and feel. The skin can also define images for most controls, to further enhance the look and feel of forms.

JADE provides a collection of skins for the JADE development environment and a global collection that contains any user-defined skins for all schemas.

The Erewhon **Administration** application provides a Skins menu that is populated dynamically with the names of all skins that are present in the system. With this menu, users can select the skin they want applied to the application. To see how this is implemented, see the **zSetupSkinSelectMenu** and **mnuSkin_click** methods on the **FormAdminMdi** class in **ErewhonInvestmentsViewSchema**.