Development of Web-Based Market Simulations for Pedagogic Use in Economics

Honors Thesis
************************
PASS WITH DISTINCTION

UH 451
Honors Senior Thesis Project
Brian Ironside
10210914
February 18, 2002
TO THE UNIVERSITY HONORS COLLEGE:

As faculty advisor for Brian Ironside

I have read this paper and find it satisfactory.

[Signature]
Faculty Advisor

2-19-02
Date
Table Of Contents

Introduction 1
The Limitations of Traditional Educational classroom activities 2
The Problem with Current Software Alternatives 4
The Advantages of DAMS 8
The Shortcomings of DAMS 12
Conclusions 14
List of Consulted Works 15
Appendix A
  User Manual for DAMS 16
  Administrator Manual for DAMS 20
Appendix B
  High Level Technical Description of DAMS Software Modules 23
Appendix C
  Source Code and Subsystem Flow Diagrams 31
Introduction

Education, like any other area of human endeavour, must change with time and technology to cope with new opportunities and challenges. Over time students change, their numbers increase, and the material they learn is in constant flux, inundated with new knowledge yielded by academia’s relentless march into the undiscovered and ill-understood. The tools available to instructors and students rotate in and out of favor, from blackboard to whiteboard to the Internet. Even the classroom is no longer a necessity in education. New communications technologies allow students and instructors to meet without having to ever set foot in the same city or country, let alone classroom. Students can seek degrees via all nature of new media, more so now than ever before with the advent of the Internet. The educators who supply such students with knowledge need new tools for new educational formats. This need provided the inspiration for the Double Auction Market Simulation software development project.

The DAMS (Double Auction Market Simulation) project is the result of the demand for additional online educational resources. It is the electronic incarnation of an educational class activity no longer practicable in its traditional form for large classes. The original Double Auction Market activity helped clarify the theory behind the function of a double auction market. In microeconomic theory, the price at which goods are sold in a free market, and the quantity sold of any good, can be predicted by balancing the supply of that good against the demand for it. The quantity sold and it’s price when supply and demand are in balance are called the equilibrium price and quantity, respectively. A Double Auction Market is simply a fancy name attached to this sort of open market interaction, where buyers and sellers haggle down to an equilibrium
price. The original Double Auction Market class activity helped demonstrate this concept by splitting the classroom into buyers and sellers in a market, and the students learned about the interaction of supply and demand in markets first hand. The students would attempt to maximize their profits by striking profitable bargains with other students, eventually haggling down to the equilibrium price. After all transactions were concluded, the resulting data was collected by the professor or teaching assistant. This data was compiled and used to demonstrate the effectiveness of the supply and demand framework of basic economic theory in predicting equilibrium outcomes in a double auction market. In a class of thirty students, an instructor could expect three rounds of interaction in a single class period.

Current introductory economics classes contain nearly two hundred students, rendering such in class activities impractical. Additionally, the Department of Economics teaches sections of introductory macroeconomics and microeconomics via the Internet, which removes such class activities from the educator's tool kit, by removing the students from the classroom. The DAMS project attempts to resolve these and other issues that arise in teaching large courses in non-traditional venues. This paper discusses some of the issues which have made traditional class activities in economics education ineffective, and how and to what extent the DAMS project dealt with these challenges.

The Limitations of Traditional Educational Classroom Activities.

Compared to the class sizes for which the original Double Auction Market activities were intended, current class sizes are enormous. Even when lectures are split into lab sections for part of the week, these sub-sections easily exceed the capacity of the Double Auction Market. The time the students need to finish their market interactions
and the amount of calculation the instructor must perform render the activity impractical. The educational environment for which the Double Auction Market was designed simply doesn’t exist any longer in the Economics Department at Washington State University.

Even in small classes, the original Double Auction Market was not terribly robust, and it was difficult to include more than the most basic aspects of microeconomic theory. Attempts to extend the amount of economics theory that could be represented and taught with this activity would quickly render it impractical from an organizational and computational standpoint. The simple addition of quotas to the market, or price controls would significantly increase the amount of time students need to play a round in the game. Including diminishing returns to scale would significantly increase the calculation required on the part of the instructor to accurately predict market outcomes. The activity allowed for basic supply and demand curve interaction, with shifts in both curves between market rounds. Not much else was involved, or possible.

An instructor needs a spare lecture hour in order to make use of the Double Auction Market activity. Often, this is something an instructors doesn’t feel they can surrender. Many lecturers are already forced to pick and choose amongst topics to cover in a introductory economics course. It’s just not possible to cover everything in a single semester. Thus, spending an entire class period with a class activity is something an instructor may be reluctant to do, despite the potential of the activity to help clarify the material they teach. Removing the activity from the classroom would allow instructors to have the best of both worlds; they could use market simulations in their curriculum without giving up precious lecture time.
As new educational formats emerge and are increasingly capitalized on, the traditional Double Auction Market has become impossible to apply. The online sections of introductory economics exist entirely without classrooms, with the exception of tests, and thus provide no opportunity for in-class activities. Distance learning courses via teleconference offer no improvement. In all situations where a lecture lacks a common, regular, physical meeting point, an alternative to the traditional Double Auction Market is needed for the students to gain first hand experience of basic market function.

An Internet based version of the Double Auction Market could alleviate many of these problems, returning the activity to the set of tools at an instructor's disposal when teaching microeconomics. Of course, if other feasible software alternative already existed, there would be little reason to develop the DAMS software. Unfortunately, searches for existing software resources to replace the traditional Double Auction Market bore little fruit.

The Problem with Current Software Alternatives

Before initiating the DAMS project, existing web-based activities were researched to ensure that the project wasn't duplicating existing work, and to see if similar work existed upon which the DAMS software could be built. The idea of moving old activities and educational resources online is not a novelty, but current software solutions are few and far between, and leave a number of traits to be desired. Though no search for such materials can claim to be authoritative due to the mass of information available on the World Wide Web, the search for web-based solutions that closely resembled the Double Auction Market activity yielded few results. Those resources that resulted did not
sufficiently resemble the double auction market that the DAMS project replaces, and had their own share of entirely novel shortcomings besides.

There are few general market simulations available online, let alone DAMS-style, freely or otherwise. Not many economists relish the idea of programming web applications, for educational or any other purposes. Though web-based programs and materials are much discussed among economics professors, usually such conversations are followed by wistful remarks about insufficient program development expertise or a lack of free time for pursuing the development of educational software. Many non-computerized sources can be found on any number of economics homepages, but little software. Some economics homepages have over one hundred posted non-computerized games, all freely available for classroom use, covering a wealth of topics. (9) Computer simulations for use in teaching economics are not plentiful, and those that exist are generally only commercially available. In an educational system already burdened by high textbook expenses, such additional costs are, at best, frowned upon.

Many of the current online resources available are extremely complex, and not well suited for use in undergraduate economics courses. An excellent example of this exists at ComLabGames, a web-site offering market simulation games, operating out of Carnegie Mellon University. (12) The software offered on this web site, though robust and flexible, would take quite a bit of effort to learn and use from an administration’s standpoint. Trying to get an entire class of two hundred some students logged in and playing would result in a logistical nightmare, as the player instructions are also intractably complex. Without technical staff to support game set-up (and a substantial budget to support the technical staff), or in classes with more students than the instructor
can personally assist, this market simulation is just not practical, despite its impressive technical merits. Unless the instructor and students were technically gifted, a curriculum would have to center itself around the software to make the investment of time worthwhile.

In addition to difficulty of use, for the purposes of demonstrating the workings of a simple Double Auction Market model, such a complex and thorough handling of the material is too robust; it's overkill. Games like EcoSim (11), the market simulation of ComLabGames (12), or other commercially available software packages, do more than is asked for or needed in a normal educational setting. Time and effort are wasted with unnecessary detail and complexity, in order to give students first hand experience with a concept that is not complex enough to warrant the use of such extensive software packages.

Many other existing online resources require all participants to be present online at the same time, and often in the same place. The World Game of Economics is a good example of one such game. (14) It doesn't even appear capable of running across a network, and thus requires all players to be physically present to participate. Such formats are unacceptable due to the large class sizes for which the DAMS project is designed. Groups of two hundred students cannot effectively meet online simultaneously. Though the students in theory could be collected in one place for this purpose, with a group of this size, no computer lab or similar computing facility has the capacity to host such a market simulation.

Another major issue with existing market simulations is portability. Even if an acceptable simulation exists, if it is computer platform dependent, then it will be of little
use to other educators. One of the most impressive educational software packages for economics and business education, Zapitalism, suffers from a complete dependence on the Microsoft Windows Operating system. (13) In the process of researching existing software for DAMS project, most sources found depended too greatly on a particular computer infrastructure, very often Microsoft Windows. Some programs only worked properly under even more specific conditions, such as only on a local area network. Both of these shortcomings prevent such software from helping educators whose information infrastructure differs from the narrow requirements of the software.

Though some commercial market simulations exist, the prices are prohibitory, and the programs are inflexible. Many commercial software packages seemed "sugar-coated", targeted more towards impressive graphics and undergraduate palatability than towards effective education. "Zapitalism", a market simulation strategy game, is an excellent example of a game with superb graphics, sound economic theory, and wonderful game play. (13) Despite it's many advantages, it is not focused purely on teaching. Though the economic theory is certainly present, and the software was developed with an education focus in mind, gaining understanding of economic principles is only one of many goals of the software. Zapitalism is a thoroughly commercial product, developed to entertain the computer gaming community as much as to aid the educational community. The DAMS project was much less expensive to develop, and could be tailored to meet the needs of the Economics Department at Washington State University. By developing the DAMS software in-house, the department had the flexibility to target it more towards effective representation of theory than accumulation of profit.
In the end there were simply very few electronic resources that closely matched the Double Auction Market activity. One can find many stock market simulations, macroeconomic simulations, and a plethora of game theory software engines online. But web applications that simulate the workings of a Double Auction Market at the level the Economics Department at Washington State University requires don’t exist, or could not be feasibly used. This fact, in addition to the many shortcomings of other software packages, contributed to the final decision to develop the DAMS software from scratch.

The Advantages Of DAMS

The DAMS software is a web-based program, essentially a series of linked web pages. Which web pages are displayed, when, and how, is determined by programs running on the web server. More specifically, the DAMS software is written in Hypertext Markup Language (HTML), with program control provided almost entirely by server-side VBScripts. Several tables in a Microsoft Access database provide data storage for the program, and the software accesses this data through ActiveX Data Objects (ADOs). Active Server Page (ASP) objects provide the foundation for the entire application. Though the preceding sentences surely have very little meaning for anyone who lacks an extensive technical background, a discussion of the tools used in programming the DAMS software is key to understanding how it solves the problems it was created to alleviate. These elements will be further explained as required.

The DAMS software emulates the original Double Auction Market activity. Players are divided into buyers and seller as they login to the web-application, and the software determines at which prices they will be able to make a profit. The player can conduct transactions via the main market display page. All results of transactions are
recorded in the application database, and information about the state of the market and
the prevailing prices are available to instructors and students alike at the press of a
hyperlink. The DAMS software also extends the toolset of the instructor with price and
time restriction options, and formats information about the activities of students for ease
of grading.

The fundamental problem that the DAMS software solves is the ineffectiveness of
traditional in-class activities in lectures that continue to balloon in size. The DAMS
software is specifically designed to work well with large groups; indeed, it depends upon
a large number of players to function properly. Using a simple web-based application to
implement the DAMS software, as opposed to a more involved network format or a
program based on a single machine, allows for computationally simple interaction
between large numbers of players. A web-based format also avoids the large amounts of
code needed to handle player interaction in network-based software. Not entirely
serendipitously, the DAMS project also solves the problem of lost curriculum options in
the web-based microeconomics and macroeconomics course taught by the Department of
Economics at Washington State University. Once the software is on the Web, it can be
just as easily integrated into other online resources and curricula, as into regular lectures.

The ability to run asynchronously is another advantage of the DAMS software.
As previously discussed, many software packages require all players to be present in the
simulation simultaneously, which isn’t possible for a large scale Double Auction Market.
Support for asynchronous participation is a requirement for any serious web-based
simulation designed for large groups of participants. The DAMS software uses an e-bay
(15) inspired format, allowing students to make and accept offers and bids without ever
once being simultaneously online with one of their peers. The players can login to the
market at intervals which suit their schedules, view the results of previous postings, and
alter their offers in the market accordingly, or accept other offers that have been posted in
the mean time. The entire simulation could function perfectly well without having any
two users simultaneously online.

Once the problem of class size is solved, the Internet format allows a wealth of
new options to be built into the market simulation. Organizational limitations on the
complexity of the simulation are resolved by making it asynchronous, and the
computational difficulties are delegated to a microprocessor, which will beat nearly any
instructor hands-down in a contest of calculation speed. With the two major limitations of
organization and calculation resolved, the DAMS software is open to the addition of new
elements of microeconomic theory beyond the original scope of the real-world Double
Auction Market. Price restrictions, quotas, diminishing returns to scale, decreasing
marginal utility, amongst many other aspects of market interaction can be added without
fear of over-complicating organization or computation of the simulation. Additional
graphical representation of data can also be made available through java applets. Though
the current DAMS software only implements a few of these potential options, due to it’s
highly modular structure, these additional pieces of functionality could be integrated at a
future date with a minimum of programming effort. In keeping with the DAMS
software’s original goals, any of these extra options can be disabled at the instructor’s
leisure, to allow the instructor to choose the options that best compliment the curriculum.

The DAMS software is simple to comprehend and use. The user is not required to
install any additional software to use the program; all they need is a java-enabled web
browser, something which is common to nearly every Internet-connected computer. Though the program consists of over twenty separate display and processing modules, and several thousand lines of code, the user interface for market players is confined to three web pages. All market interaction is conducted from a single market display and control screen, and all relevant information concerning the market is contained in a single frame of that control screen. The user instructions sum up all of the information needed to play the game in less than four pages. For administrators the program is a little more complex. DAMS can be used by adding the names of the players to the user database, a task no more complex than adding information to a Microsoft Excel spreadsheet. Additional software functionality can be used as the administrator learns the software. The DAMS software is simple enough to be used by many with a minimum of effort, while robust enough to allow more complex market interactions.

One of the reasons DAMS was developed, as opposed to purchasing a commercially available market simulation was code control. By owning the source code used to create the DAMS software, the Economics department can always rework it to fit the needs of a specific curriculum. Instead of conforming the curriculum to match the style of an out-of-the-box commercially available market simulation, modules within the DAMS software can be switched out, and source code tweaked to match the educational goals of the instructor. Additionally, should bugs arise in the software, they can be remedied immediately and directly, as opposed to waiting for software fixes that may or may not be available in time to be meaningful for restoring aborted lesson plans.
The Shortcomings Of DAMS

DAMS has its fair share of disadvantages. In addressing portability issues, the DAMS software is not perfectly flexible. The Active Server Page basis, use of Microsoft Access databases, and reliance on the VBScript scripting language all restrict it to Microsoft Servers. This is the only real restriction on the software’s use. Though the DAMS software is entirely Microsoft platform dependant at the server, the software is accessible via the Internet from any platform. Any java-enabled web browser can interact with the program without encountering platform compatibility issues. Even though the program uses Microsoft dependant components, once the program sends information from the server to the client computer, all interaction takes place through HTML, which is common to all web content, and Java, rendering the program nearly universally readable on the client side. Thus, the software restricts the platform choice of only the server, and not the two hundred or more that will interact with it. Though not perfect, this allows for extreme flexibility in user platform choice.

The choice of software tools used in the development of the DAMS software is less than perfectly efficient. Specifically, database software other than Microsoft Access would certainly add processing power and speed to the DAMS software. The software design is not always optimally efficient. More development time could be put into resolving these issues, and could probably marginally speed up the run time of a few code modules. Whether such efforts are worthwhile is debatable, however. The DAMS software could easily handle several thousand students with no noticeable slowdown in processing time for any given task. The testing done on the DAMS software as of this paper has not shown any problems related to processing efficiency. Better, faster, and
more robust Database software is certainly available, but as the current database software seems capable of handling all of DAMS data storage and access needs, no convincing argument for the replacement of that software has presented itself. DAMS is capable of handling orders of magnitude more students than are currently intended to use it, and so concerns of additional efficiency given the expected context of its use do not seem very pressing.

Visually, DAMS doesn't measure up to most of its contemporaries. Even the simplest graphical interfaces, such as though used by the World Game of Economics, (14) are still aesthetically superior to the DAMS interface. The graphical difference is vast between DAMS and more visually appealing simulations such as Zapitalism, or any of the other games available from the Lavamind corporation. (13) This is a minor problem, however, given the advantages of modest aesthetics in a web-based application. Since all information must be sent from the server hosting the program to each client each time the client requests a new page, advanced graphics would quickly overwhelm a slow Internet connection. If involved enough, advanced graphics can even tax faster DSL and cable modem connections. A more engaging user interface could also potentially distract students from the theories they are supposed to learn through the DAMS software.

DAMS does not possess the extensive functionality of many commercially available programs. In its current form, the DAMS software is only useful for teaching a small part of microeconomic theory, dealing with supply and demand interaction in a microeconomic context. This simplicity is intentional, to avoid complicating the concepts DAMS demonstrates with extraneous functionality. Though further development of the
DAMS software could easily extend the options available to an instructor, every added option is another option the instructor, and potentially the students, will need to learn to use the software. One of the major design goals of DAMS is ease of use and speed of comprehension. Neither instructors nor students have large amounts of time to waste learning new software. For an educational software package to help in teaching any subject, it must not be so time consuming to use that it detracts from the full and effective teaching of the concepts central to a curriculum. Even the few pieces of additional functionality the DAMS offers an instructor are completely optional, and instructors can use the software while remaining blissfully ignorant of these software options. Though DAMS can only be used to teach a very limited set of concepts, its simplicity allows its use with a minimum of additional strain to instructors and students.

Conclusions

Though it has not yet been used as an educational tool in an economics class, the development of the DAMS software has been successful. The Double Auction Market has been transferred to an electronic format, and has kept with the spirit of the original activity and the implementation suggestions (2, 3) made in the software specification provided by Dr. Wayne Joerding at the beginning of the project. The DAMS software allows the continued use of class activities rendered impractical by changing educational venues and formats, without the increased overhead normally associated with educational software packages.
List Of Consulted Works


2) Dr. Wayne Joerding. Negotiated double Auction Market, Part I & II (Class Activities).


Appendix A: DAMS User And Administrator Manuals
User directions for the Double Auction Market Simulation

Logging in
When you first get to the login page you will see our lovely sparkling welcome banner. Underneath you will see two text boxes (see picture below) where you will enter your name exactly as it appears on the class roster. Normally this is last name, first name, possibly with the middle initial depending on how your name is registered with the university office of admissions. If you’re not sure, click on the link marked “View Current Registered Players.” The names of all registered players are listed, in the format the login requires. In the space below that, you will enter your student ID number. Next, click on the submit button or press enter. This will log you into the Double Auction Market Simulation.

You will be assigned buyer or seller status as you log in. Your status will be displayed at the top of your screen. The screen should look something like the picture below.
Double Auction Market Simulation

Welcome, Pam Anderson. You are a Buyer in the current market

Your name will be here

Your status as a Buyer or Seller will be displayed here

Buyer's Information

Change or Post Bid: [Input Field]

Current Bid: $0.00
Status of Bid: None Posted

Refresh Market Display
View Plot of Recent Prices
Logout

Current Buyers

<table>
<thead>
<tr>
<th>Buyer ID</th>
<th>Listed Bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>18976534</td>
<td>$5.50</td>
</tr>
<tr>
<td>98125467</td>
<td>$2.50</td>
</tr>
</tbody>
</table>

Current Sellers

<table>
<thead>
<tr>
<th>Seller ID</th>
<th>Listed Price</th>
<th>Accept?</th>
</tr>
</thead>
<tbody>
<tr>
<td>111111111</td>
<td>$6.00</td>
<td>Accept</td>
</tr>
<tr>
<td>98767890</td>
<td>$2.50</td>
<td>Accept</td>
</tr>
</tbody>
</table>

Buying and Selling

There will be two frames, one on the left and one right of your screen, labeled Current Buyers and Current Sellers. The Current Buyers frame lists all buyers who have posted bids, and how much they are willing to pay. Likewise, the Current Sellers frame lists all of the current offers listed for people who want to sell and how much they are willing to sell for.

Note that these bids are current as of when you first logged in. Other people may be adding new bids while you are looking at this screen. To get the most current buyer and seller information you will have to refresh the market information. You can do this by clicking on the Refresh Market Display link or by hitting the refresh button in your browser.
Recent Prices
If you want to see the latest sale prices click on the View Plot of Recent Prices link.

Reservation Price and Production Cost
Make note of your Reservation Price if you are a buyer. As a buyer, your goal is to purchase from the seller whose offer price is lower than your Reservation Price. As a buyer, you must accept an offer LESS than your reservation price to earn Consumer Surplus. Consumer surplus is a measure of your success as a buyer. The less you pay, the greater this number will be. If you are a seller you will be assigned a Production Cost. You must sell for MORE than your Production Cost to earn a Profit if you are seller. The higher the offer you accept from a buyer over and above your production cost, the greater profit you will earn.

Accepting Standing Offers
At this stage you should review the bids offered and see if you want to accept one. If you find one that you like, click on the Accept button next to that bid. If not, do not be alarmed; you can always log off and log back on later when more people have posted bids, when your chances of finding an acceptable offer are higher.

Posting Offers or Bids
If no standing offers are suitable for your production cost or reservation price, you can post your own offer, and hope someone else accepts it later. If you wish to change your offer, or have made an error in your post, just re-enter the offer you want, and the new offer will replace the old. Remember, if you are a buyer, your offer should always be lower than your reservation price, and if you are a seller, your offer should always be higher than your production cost. You can check the
status of your offer by scrolling through the current sellers or current buyers. Your offer will be highlighted in blue.

If your offer is accepted, it will be removed from the current buyer or current seller listings. The next time you log in your offer status will have been changed to “accepted” and the results of the transaction will be displayed in the buyer/seller information portion of the your market display screen.

**Price Floors and Ceilings**
In some market simulations, your instructor may impose price floors or ceilings. These will be displayed in your Buyer/Seller Information frames, if they exist. In the event of a price floor, no offers may be posted below the amount of the price floor. If there is a price ceiling, no offer above the price ceiling may be posted. If you attempt to do either of these things, the program will give you an error message, and return you to the market simulation without posting your bid or offer.

**Viewing Your Profit Or Consumer Surplus**
Once you accept an offer your profit or consumer surplus earned will be displayed below your production cost or reservation price.

Once you have accepted an offer, you will not be able to change the offer you accepted, or accept any further offers. You will still be able to log on and view the market and your own player status, but you will not be able to perform any further market interactions.

**Situations Where You Can’t Make A Profit**
It may be the case that you will not be able to make a profit because of the way your player account was set up. In fact about half of all users will be in this situation. If your production cost is too high, or your reservation price too low, it may not be possible to make a profitable deal in the market. In this case, the best thing you can do is not make any deal at all. Your profit or consumer surplus will be zero, but in this case, that's the best you can do.

**Logging Out**
Once you are finished playing, either for the entire game, or just until the next time you want to check the status of the market, click on the logout link to exit the game. Once you've logged out, you can always log back in by returning to the login page or clicking on the login link on the logout page.
Administrator directions for the Double Auction Market Simulation (DAMS)

Game Setup

Initializing Players

The administrator must add all persons who interact with the website to the game database, either as a normal player or administrator. The Microsoft Access file found in the fpdb directory (fpdb/UserDB.mdb) contains the database for the simulated market. The UserDB.mdb database contains three tables, MarketDB, UserDB, and ScheduleDB. Open the table named “UserDB”.

For normal players, you only need to complete two columns of the UserDB table. Add the player’s name to the field “strUser”, and the player’s student ID number (or other suitable unique identification number) to the “intStudentID” field. The simulated market software will add and manipulate all other information during the course of the game.

Use exactly the same procedure to initialize a login account with administrator rights, except type in the word “Admin” in the field “strUserType”.

Default Market Settings

The market has built-in default settings, so play can begin once the players’ information has been added. The default market has no price floors or ceilings. The demand curve has a minimum value of 2, and a maximum value of 14, and a slope of negative 1. The supply curve has a minimum value of 2, a maximum value of 14, and a slope of positive 1. These settings may be altered in the current market parameters and control options screen, described below.

The Administrator Options Screen

After logging in, an administrator will first encounter the Administrators Options screen. This web page provides a brief description of each option button. You’ll find a more complete description below.

Viewing the Market Simulation

This button will take you to the Market simulation page where the players interact with each other, posting offers, making sales, etc. You can interact with this screen in the same way as any other player, with the exception that an administrator cannot post or accept offers. If you want to play the game like a normal player, create an additional entry in the UserDB table with an alias name and arbitrary ID number. The options available in the market screen are detailed in the Student Manual for DAMS.

Resetting the Market

Pressing the Reset Market button will do a number of things. First of all, it erases all game data except for the names and ID numbers of the players, wiping away any actions any player has made in the market. All players start over with a clean slate, and will be re-initialized with new market roles and values when they next log on.
Second, resetting the market will reset the supply and demand curves to their default settings, and erase any price floors or ceilings that may exist.

**Viewing the Current Market Status**

Pressing the view current market button will open a new window containing all of the vital information about all registered users. Name, ID number, market role, and profit or consumer surplus will all be displayed. The market role will come up as Buyer or Seller, unless the player has never logged on to the website, in which case it will show “unassigned.” The status of the offer field will be “none posted” if the player has posted no offer, “standing” if the offer is still standing, or “accepted” if the player’s offer has been accepted. If the offer has been accepted, the profit or consumer surplus for that player will also be displayed.

The layout of this web page is kept intentionally simple for ease and efficiency of printing. To print a copy of the web page, simply select file->print in your web browser. You can also copy and paste this page into a spreadsheet for further analysis.

**Viewing the Market Supply and Demand Schedules**

This button will take you to a screen that displays the demand and supply schedules for the market. The various price levels in the market, as well as how many agents (buyers and sellers) will be willing to buy or sell at each price, are displayed here. This data can be used to predict the equilibrium price and quantity in the market. If you wish to generate graphs from the data, it can be highlighted and copied into a spreadsheet program like excel to create graphs.

**Alter the Market**

Pressing the Change Market Parameters button will take you to the Current Market Parameters and Control Options page. Here you can view the current market settings such as supply and demand curve values, and any price ceilings or floors that may exist.

**Current Market Parameters**

This section of the page will display current price floors and ceilings, and the demand and supply curve parameters. The demand and supply curves are broken into three component parts: the minimum value, the maximum value, and the increment value. The increment value equals the step size of the curve, and the minimum and maximum values determine the smallest and largest reservation price/production cost that can be assigned to a player.

**Adding Price Ceilings and Floors with Market Restriction Options**

Adding price floors and ceilings to the market is very straightforward; simply click which you want to add, a floor or a ceiling, type in the amount, and press submit. The changes will show up immediately in the current market parameters section of the screen.

To remove either the price floor or price ceiling, just type “remove” in the input box, select which restriction you want to remove (floor or ceiling), and press submit. The changes will show up immediately in the current market parameters.
Shifting the Supply and Demand Curves

There are two ways to change the supply and demand curves; an easy way and a hard way. The advanced options give you much more control over the demand and supply curves, but require a more intimate understanding of how the program works to use properly. The simple shifts are not so involved, and fairly self-explanatory.

Simple Shifts

Simple shifts up or down in the supply and demand curves can be accomplished in this portion of the screen. The shifts here will only alter the minimum and maximum values of the demand or supply curve (or both). The increments will remain the same. Just input the amount you want to shift the curve by, and press the submit button. The results should show up immediately. Negative numbers will shift the curve down.

Advanced Shifts

For more control over the demand and supply curves, you can use the advanced supply and demand options. You have complete control over all of the values that control the supply and demand curves. The maximum and minimum values should be fairly self-explanatory; these are the highest and lowest production cost/reservation prices that a seller/buyer can have.

The increment value will determine the slope of the supply or demand curve. Even though the slope of the demand curve is negative, the increment value must be positive. The increment value cannot be greater than the difference in the minimum and maximum values for that curve, or the resulting curve will be flat. Even if the value meets this requirement, larger values will result in very few entries in demand and supply schedule printouts. This is due to the way the program initializes the players the first time they log in.

Once you’ve input all of the values you want to change, press submit, and the changes in the supply and demand curves will take affect immediately.

Returning to the Administrative Options Page

Once you’ve finished in the market parameters and control options page, you can return to the main admin options page by clicking the link at the bottom of the page.

Logging out

You can log out from either the Admin options page, or the market display page. On the admin options page, you can log out by pressing the logout button at the bottom of the page. From the market display page, you can log out by clicking the log out link in the middle of the user information portion of the market display page.
Appendix B: High Level Technical Description of DAMS Software Modules
**DAMS Technical Description**

**Data Sources**

The DAMS (Double Auction Market Simulation) web application uses a single Microsoft Access database stored in the fpdb directory, UserDB.mdb, containing three tables. These tables store DAMS’ data and market settings. The tables are described below.

**UserDB**

The userDB must be initialized before the program can function. It must contain the Name and Student ID number of every player in the game. Additionally, it must contain the numeric password (in the StudentID field) of any administrators, as well as the string “Admin” in the StrUserType field.

- **StrUser** => This string field contains the name of a player or administrator, and is used at login as the user name.
- **IntStudentID** => This integer field is the student ID # of the student playing the game, or an appropriate number for administrators and players who are not students. IntStudentID is the primary key for this table.
- **CurReservationOrCost** => The reservation price or production cost of a player is stored here as a currency value.
- **CurUtilityOrProfit** => If any profit is made by a player, that quantity will be stored here as a currency value.
- **BlnPlayStatus** => The Boolean value in this field determines whether or not a player can continue interacting with the market actively. It is set to false when a player can no longer play.
- **StrUserType** => For players, this value will be set by the program upon their first logon. The program will assign a “Buyer” or “Seller” value to this field if it is empty the first time a user logs on. If the person is an administrator, this field must be set to “Admin” manually before the administrator logs on to the website for the first time.

**MarketDB**

At the start of the game, this table will be empty.

- **IntOfferer** => Whenever a player posts an offer in the market, their ID number is recorded in this integer field.
- **IntAccepter** => Whenever an offer is accepted, the student ID of the accepter is recorded in this integer field.
- **CurOffer** => The current offer posted by the IntOfferer is recorded in this currency field.
- **IntTransaction** => This integer field records the order in which all transactions in the market were finished.

**ScheduleDB**

DAMS stores its basic market settings here. This table links to no other, and only ever has one row if DAMS is functioning properly. At start-up, all fields contain a default value, which will be described below.
CurDMax => This currency field contains the maximum value for the demand curve; that is, the maximum reservation price that any buyer in the market can be assigned. Its default value is 14.

CurDMin => This currency field contains the minimum value for the demand curve; that is, the minimum reservation price that any buyer in the market can be assigned. Its default value is 2.

CurDInc => This currency field contains the increment value for the demand curve. Every time a new buyer is initialized, the reservation price assigned is increased by this amount, until the CurDMax is reached. Then the reservation price cycles back down to the CurDMin. This process will be explained in greater detail later. The default value is 1.

CurSMax => This currency field contains the maximum value for the supply curve; that is, the maximum production cost that any seller in the market can be assigned. Its default value is 14.

CurSMin => This currency field contains the minimum value for the supply curve; that is, the minimum production cost that any seller in the market can be assigned. Its default value is 2.

CurSInc => This currency field contains the increment value for the supply curve. Every time a new seller is initialized, the production cost assigned is increased by this amount, until the CurSMax is reached. Then the production cost cycles back down to the CurSMin. This process will be explained in greater detail later. The default value is 1.

CurPFloor => If a price floor exists in the market, its value will be stored in this currency field. If there is no price floor, the value stored at its default value of -10000.

CurPCeiling => If a price ceiling exists in the market, its value will be stored in this currency field. If there is no price ceiling, the value stored at its default value of -10000.

Contents and Function of Application and Session Objects
Application Object
Data Description
intSellers => The number of sellers initialized in the market.
intBuyers => The number buyers initialized in the market.
intTotalSales => Total number of transactions executed in the market.
curAvgPrice => The average price at which the transactions in the market take place.
intRecordsEffected => Used for database access upon object startup.
DBInit => The connection object used to initialize market variables on object startup.
Record => The recordset object used to initialize market variables on object startup.
StrSellersQuery => The SQL query that counts the number of sellers in the market.
StrBuyersQuery => The SQL query that counts the number of buyers in the market.
StrTotalSalesQuery => The SQL query that counts total transactions.
StrCurrentMarketQuery => The SQL statement that fetches the rest of the important data.
StrMarketConnectionString => The connection string used to connect to the database.

Other Functions
The application object holds global data for user initialization. The intSellers and intBuyers are updated through program flow after the initial lookup in the database.
Session Object

Data Description

IntUserID => Student ID and password
StrUserType => User's player type
CurPriceOrCost => The reservation price or production cost of the user
CurUtilityOrProfit => If the user has made a sale, the profit/utility is stored here
CurOfferOrBid => If the user has posted a bid, that amount is stored here
BlnPlayStatus => True by default, false if the user can no longer interact with the market.
StrResponse => Used to communicate with the user, reporting errors and other important information.
StrStartPage => The page the session should start on.
StrStart => The first page the user tries to access. Should match StrStartPage.
User => User Name
Password => User password, normally the studentID

Other Functions

In addition to storing all user information, there is a small if statement in the Session_OnStart function that redirects the user to the Login page at startup if they tried to go to a different page first. All user information is stored and updated in this object as the user navigates DAMS.

List and Description of Component Pages

Login Subsystem

Login.asp

Links From => Logout.asp, Login.asp
Linked To => Switch.asp, Login.asp, CurrentUserList.asp
Visible? => Yes
Database Access => None
Description => This page links to the current user list to allow new users to see how the program expects them to type their login name. Login occurs here. After entering a user name and password, the page checks these two fields for basic validity, and then submits them to switch.asp for final processing.

CurrentUserList.asp

Links From => Login.asp
Linked To => None
Visible? => Yes
Database Access => UserDB
Description => This page displays a table containing all usernames of users registered to play in the game.

Switch.asp

Links From => Login.asp
Linked To => Login.asp, Admin.asp, MarketDisplay.asp
Visible? => No. This page only performs internal processing
Database Access => UserDB, ScheduleDB
Description => This page has two functions. It checks for the existence of the user and their user type, and redirects them as appropriate. If the user has not yet been initialized, this occurs here when the user first logs in.

The initialization process is somewhat involved. First, the program checks the number of buyers and sellers in the market, as recorded in the application object, and assigns a user type from whichever type is least represented, i.e. if there are more buyers in the market, new users will be initialized as sellers. This information is recorded in the session object.

The most involved part of initialization is calculating and setting the reservation prices and production costs, given current market parameters. The program queries the ScheduleDB to retrieve the current maximum, minimum, and increment values for the reservation price (demand curve) and the production cost (sellers). With the exception of the curve used for calculating values, the initialization process is identical for both user types.

The program begins a loop that will execute a number of times equal to the number of users of the same user type as the user in initialization. The loop begins with the minimum value for the demand or supply curve, depending on the user type. At each loop iteration, this amount increases by the increment value. If the value reaches or breaches the maximum value, it is reset to the minimum value. The value remaining at the end of the loop becomes the production cost or reservation price for the new user. The program records this in the session object, and updates all new user data in the database. The new user then proceeds to MarketDisplay.asp.

Admin Subsystem

Admin.asp
Links From => Login.asp, MarketDisplay.asp, Supply and Demand.asp,
ClearMarket.asp
Linked To => Logout.asp, MarketDisplay.asp, Supply and Demand.asp,
CurrentMarketInfo.asp, Market Schedules.asp, ClearConfirm.asp
Visible? => Yes
Database Access => None
Description => This page contains all of the administrator’s game interaction options. All the data display and game control options are reached from this page.

CurrentMarketInfo.asp
Links From => Admin.asp
Linked To => None
Visible? => Yes
Database Access => UserDB, MarketDB
Description => This page displays vital information about the players in the simulation, their progress in the simulation, and degrees of success. The display is kept simple for ease of printing.

Market Schedules.asp
Links From => Admin.asp
Linked To => None
Visible? => Yes
Database Access => UserDB, ScheduleDB
Description => This page reads in data from the market settings and current users, and calculates the current demand and supply schedules in the simulation based on these data. The page displays price and the number of users willing to buy or sell at that price for the demand and supply curves. The display is kept simple for ease of printing.

**ClearConfirm.htm**

Links From => Admin.asp
Linked To => ClearMarket.asp, Admin.asp
Visible? => Only a JavaScript Confirm Box is displayed here.
Database Access => None
Description => This page displays a simple JavaScript Confirm Box, and redirects to the real clear market page, or back to admin. It is included as a safety feature, in case someone accidentally clicks on the clear market option.

**ClearMarket.asp**

Links From => ClearConfirm.htm
Linked To => Admin.asp
Visible? => No. This page is purely an internal processing page.
Database Access => UserDB, MarketDB, ScheduleDB
Description => ClearMarket.asp erases the MarketDB, and clears all data from the UserDB except for user names and ID numbers, and admin user types. It resets ScheduleDB to it’s default settings; all minimum values are 2, all maximum values are 14, all increment values are 1, and all market restrictions are -10000.

**Market Parameter Control Subsystem**

**Demand and Supply.asp**

Links From => Admin.asp, ChangeSchedule.asp
Linked To => ChangeSchedule.asp, Admin.asp
Visible? => Yes
Database Access => ScheduleDB
Description => This is a multipurpose market control and display page. It displays the current supply and demand curve parameters, as well as the existing price floors and ceilings. Forms exist for controlling price floors and ceilings, as well as simple and advanced supply and demand curve shifts. All forms submit to ChangeSchedule.asp for processing.

**ChangeSchedule.asp**

Links From => Demand and Supply.asp, changeSchedule.asp
Linked To => Demand and Supply.asp, ChangeSchedule.asp
Visible? => No. This page only does processing
Database Access => ScheduleDB
Description => ChangeSchedule.asp does a number of things based on what parameters were passed to it by Demand and Supply.asp. The three submit buttons on the the supply and demand.asp page will determine which section of the code in ChangeSchedule.asp DAMS executes.

First, if the user submits a market restriction (price floor or ceiling), DAMS checks for a valid value and ensures that the market is working properly before writing the new restriction to the ScheduleDB. If the user issued a “remove” command, the specified restriction is set to -10000 in ScheduleDB, which signifies a lack of restriction.
If the user submitted a simple demand or supply curve shift, DAMS checks the submission for errors, and then updates the ScheduleDB with the new curve shifts. If there is an error, DAMS redirects back to Admin.asp with an error message.

The advanced supply and demand curve options are a bit more involved, but function essentially the same as the simple options. The values submitted by the user are checked for errors, and then instead of calculating shifts based on current values, the update statement simply inserts the values submitted as-is.

After any of these options executes successfully, DAMS redirects the user back to Admin.asp.

**Market Simulation Display Subsystem**

**MarketDisplay.asp**

- Links From => Admin.asp, Login.asp
- Linked To => None
- Visible? => Sort of. This page creates the frame structure for the market simulation, but the actual pages shown are contained elsewhere.
- Database Access => None
- Description => This is the frame set for the market display. No real processing occurs here.

**TopMarket.asp**

- Links From => MarketDisplay.asp (frameset super-page)
- Linked To => None
- Visible? => Yes
- Database Access => None
- Description => This page displays a simple welcome, showing the user's name and market role (buyer or seller).

**CurrentBuyers.asp**

- Links From => MarketDisplay.asp (frameset super-page)
- Linked To => MakeSale.asp
- Visible? => Yes
- Database Access => UserDB, MarketDB
- Description => This page within the market display contains all of the current offers to buy in the market, and displays the option to accept an offer if the current user is a seller who is still active in the market.

**CurrentSellers.asp**

- Links From => MarketDisplay.asp (frameset super-page)
- Linked To => MakeSale.asp
- Visible? => Yes
- Database Access => UserDB, MarketDB
- Description => This page within the market display contains all of the current offers to sell in the market, and displays the option to accept an offer if the current user is a buyer who is still active in the market.

**UserInfo.asp**

- Links From => MarketDisplay.asp (frameset super-page)
- Linked To => PostOffer.asp, Logout.asp, Graph.asp, Admin.asp
- Visible? => Yes
If the user submitted a simple demand or supply curve shift, DAMS checks the submission for errors, and then updates the ScheduleDB with the new curve shifts. If there is an error, DAMS redirects back to Admin.asp with an error message.

The advanced supply and demand curve options are a bit more involved, but function essentially the same as the simple options. The values submitted by the user are checked for errors, and then instead of calculating shifts based on current values, the update statement simply inserts the values submitted as-is.

After any of these options executes successfully, DAMS redirects the user back to Admin.asp.

**Market Simulation Display Subsystem**

**MarketDisplay.asp**
Links From => Admin.asp, Login.asp
Linked To => None
Visible? => Sort of. This page creates the frame structure for the market simulation, but the actual pages shown are contained elsewhere.
Database Access => None
Description => This is the frame set for the market display. No real processing occurs here.

**TopMarket.asp**
Links From => MarketDisplay.asp (frameset super-page)
Linked To => None
Visible? => Yes
Database Access => None
Description => This page displays a simple welcome, showing the user's name and market role (buyer or seller).

**CurrentBuyers.asp**
Links From => MarketDisplay.asp (frameset super-page)
Linked To => MakeSale.asp
Visible? => Yes
Database Access => UserDB, MarketDB
Description => This page within the market display contains all of the current offers to buy in the market, and displays the option to accept an offer if the current user is a seller who is still active in the market.

**CurrentSellers.asp**
Links From => MarketDisplay.asp (frameset super-page)
Linked To => MakeSale.asp
Visible? => Yes
Database Access => UserDB, MarketDB
Description => This page within the market display contains all of the current offers to sell in the market, and displays the option to accept an offer if the current user is a buyer who is still active in the market.

**UserInfo.asp**
Links From => MarketDisplay.asp (frameset super-page)
Linked To => PostOffer.asp, Logout.asp, Graph.asp, Admin.asp
Visible? => Yes
**Database Access** => ScheduleDB

**Description** => This page displays loads of user information, as well as market restrictions and current user status. This is the major interaction page for the user. Errors are also displayed here if the user attempts an illegal action.

**Graph.asp**

**Links From** => UserInfo.asp

**Linked To** => None

**Visible?** => Yes

**Database Access** =>

**Description** => As of this printing, the graph.asp page has not yet been implemented

**Makesale.asp**

**Links From** => CurrentBuyers.asp, CurrentSellers.asp

**Linked To** => MarketDisplay.asp

**Visible?** => No. This is a processing page.

**Database Access** => MarketDB, UserDB

**Description** => This page handles the processing of sales when a player accepts a posted offer. First, it checks to make sure the user is not an administrator, or someone who no longer has rights to interact in the market. Errors are also raised if the offer selected has already been accepted elsewhere.

If the sale is approved, DAMS combines the records in the MarketDB of the buyer and seller into one record, and assigns the new record a transaction number based on its sequence in the market. The new record contains the price at which the transaction occurred, as well as the student ID number of the seller in the intOfferer field and the ID number of the buyer in the intBuyer field. After this is complete, or an error is raised, DAMS returns to MarketDisplay.asp.

**PostOffer.asp**

**Links From** => UserInfo.asp

**Linked To** => MarketDisplay.asp

**Visible?** => No. This is a processing page.

**Database Access** => MarketDB, ScheduleDB

**Description** => The postoffer.asp page records or updates offers in the market. The first half of the page is devoted entirely to error checking, to ensure administrators don’t post, and all offers are within the bounds set by the price floors and ceilings. The offer is also checked to ensure proper data type.

After the data has passed these tests, the MarketDB is checked to see if an offer already exists for the student ID number that submitted the current offer. If an offer already exists, the record in the MarketDB is updated to the new value. If no record exists, a new record is added to the MarketDB with the current offer. After the offer is posted or an error occurs, DAMS returns to the MarketDisplay.asp page.

**Miscellaneous**

**Logout.asp**

**Links From** => Admin.asp, UserInfo.asp

**Linked To** => Login.asp

**Visible?** => Yes

**Database Access** => None
Description => This page is displayed after logout, and has a link for re-login. The user’s session is abandoned here.
Appendix C: Source Code and Subsystem Flow Diagrams
Login Subsystem

Start

Login.asp

CurrentUserList.asp

Logout.asp

Switch.asp

User DB

Schedule DB

Admin Subsystem

Market Simulation Subsystem

User DB
<%@ Language=VBScript %>
<%@Response.CacheControl = "Private" %>
<HTML>
<HEAD>
<title>List Of Current Users</title>
</HEAD>
<BODY>
<p align="center"><u><font size="5">List of Current Registered Players</font></u></p>
<%Dim DataRecord
Dim StrDataQuery
'Initialize Database objects
Set DataRecord = Server.CreateObject("ADODB.RecordSet")

'Initialize Query strings
StrDataQuery = "Select StrUser From UserDB Where strUserType = 'Buyer' OR strUserType = 'Seller'"

'Retrieve data
Set DataRecord = Connection.Execute(StrDataQuery, ,adCmdText)

'Set up the DataRecord recordset with additional market data
if not DataRecord.EOF then
    DataRecord.MoveFirst
    'loop through each record, displaying along the way
    Do
        <tr><td><%= DataRecord.Fields("strUser") %></td></tr>
    Loop until DataRecord.EOF
else
    <tr><td>No players are currently registered to play</td></tr>
end if
</table>
</BODY>
</HTML>
<@ Language=VBScript @>
<
If not Isempty(Request.Form("User")) and not IsEmpty(Request.Form("PW")) then
  If isNumeric(Request.Form("PW") then
    Session. Contents. Item("User") = Request. Form("User")
    Session. Contents. Item("PW") = Request. Form("PW")
    Response. Redirect("Switch.asp")
  else
    session. Contents. Item("strResponse") = session. Contents. Item("strStartPage") "+"The Login or Password contained an error"
  end if
end if

<html>
<head>
title=Login/title>
<meta name="Microsoft Theme" content="maize"
<body background="maizbk.jpg" bgcolor="#FFFFFF" text="#OOOOO0" link="#669999" alink="#999900" alink="#3366CC">
<font face="Arial, Helvetica" mstheme>
<p align="center"><b><font size="6">Welcome:</b></font></p>
<p align="center"><b><font face="Arial, Helvetica" size="6">Double Auction Market Simulation</font></b></p>
<p align="center"><b><font face="Arial, Helvetica" size="6">Please enter your name and WSU Student ID number and press SUBMIT:</b></font></p>
<form method="POST" name="loginForm" action="login.asp">
<table BORDER="5" CELLPADDING="5" bordercolorlight="#999900" bordercolordark="#666633">
<tr>
<td><font face="Arial, Helvetica" mstheme>
<p align="center"><b>First And Last Name:</b></p>
</font mstheme>
<td><input NAME="USER" TYPE="text" SIZE="15"></td>
</tr>
<tr>
<td><font face="Arial, Helvetica" mstheme>
<p align="right"><b>ID Number:</b></p>
</font mstheme>
<td><center><input NAME="PW" TYPE="password" SIZE="15" MAXLENGTH="132"></center></td>
</tr>
</table>
<input type="submit" value="Submit" name="Bl"></form>
<p align="center"><b><u><a href="Current.UserList.asp" target=1ttop><font color="#000000">view Current Registered Players</a></u></b> -
'Currently removed for debugging purposes
' today = dateValue("12/31/4444")
' endDate = dateValue(Application. Contents. Item("timGameEnd"))
' if endDate <= #12/31/9999# and (datediff(d, endDate, today) <= 0) then
'   <p align="center"><b><u><a href="PriceSeries.asp" target=1ttop><font color="#000000">View Market Transaction Time Series</a></u></b>
' end if
' cp align="center",<img border="0" src="falling_money.gif" width="108" height="136"></p>
<p align="center">Session. Contents. Item("strResponse") %></p>
</font mstheme></center>
</body>
</html>
Dim StrLoginQuery
Dim DBUser
Dim LoginRecord

'Here we put together the SQL Query String for fetching User data
StrLoginQuery = "Select * From UserDB Where strUser = " & Session.Contents.Item("User") & " AND intStudentID = " & Session.Contents.Item("PW")

'This creates and sets the database connection
Set DBUser = Server.CreateObject("ADODB.Connection")
DBUser.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source= " & Server.MapPath("fpdb/UserDB.mdb")
DBUser.Open

'Now we initialize the recordset that will hold the user's basic market information
Set LoginRecord = Server.CreateObject("ADODB.RecordSet")
'And then we query the database using the SQL statement defined above
LoginRecord.open StrLoginQuery, DBUser, 2, 3, 1
If not LoginRecord.EOF then
    LoginRecord.MoveFirst
End If

'This first if-statement will check if the login exists. If not the page is re-routed back to login with an error message.
If LoginRecord.EOF Then
    Session.Contents.Item("StrResponse") = "Invalid Login. Please try again or contact your instructor if problems persist."
    'Free up system resources
    DBUser.Close
    Set DBUser = Nothing
    Set LoginRecord = Nothing
    Response.Redirect "login.asp"
ElseIf (LoginRecord("strUserType") = "Admin") then
    Session.Contents.Item("IntUserID") = LoginRecord("intStudentID")
    Session.Contents.Item("StrUserType") = LoginRecord("strUserType")
    Session.Contents.Item("CurPriceOrCost") = LoginRecord("curReservationOrCost")
    Session.Contents.Item("CurUtilityOrProfit") = LoginRecord("curUtilityOrProfit")
    Session.Contents.Item("BlnPlayStatus") = LoginRecord("BlnPlayStatus")
    Dim StrCheckMarketQuery
    StrCheckMarketQuery = "Select curOffer from MarketDB where intOfferer = " & Session.Contents.Item("intUserID")
    LoginRecord.Close
    LoginRecord.open StrCheckMarketQuery, DBUser, 2, 3, 1
    if not LoginRecord.EOF then
        Session.Contents.Item("CurOfferOrBid") = LoginRecord("curOffer")
    end if
ElseIf (LoginRecord("strUserType") = "Buyer") then
    Session.Contents.Item("IntUserID") = LoginRecord("intStudentID")
    Session.Contents.Item("StrUserType") = LoginRecord("strUserType")
    Dim StrCheckMarketQuery
    StrCheckMarketQuery = "Select curOffer from MarketDB where intOfferer = " & Session.Contents.Item("intUserID")
    LoginRecord.Close
    LoginRecord.open StrCheckMarketQuery, DBUser, 2, 3, 1
    if not LoginRecord.EOF then
        Session.Contents.Item("CurOfferOrBid") = LoginRecord("curOffer")
    end if
ElseIf (LoginRecord("strUserType") = "Seller") then
    Session.Contents.Item("IntUserID") = LoginRecord("intStudentID")
    Session.Contents.Item("StrUserType") = LoginRecord("strUserType")
    Dim StrCheckMarketQuery
    StrCheckMarketQuery = "Select curOffer from MarketDB where intOfferer = " & Session.Contents.Item("intUserID")
    LoginRecord.Close
    LoginRecord.open StrCheckMarketQuery, DBUser, 2, 3, 1
    if not LoginRecord.EOF then
        Session.Contents.Item("CurOfferOrBid") = LoginRecord("curOffer")
    end if
Else
    if loginRecord("strUserType") <> "" then
        Session.Contents.Item("IntUserID") = LoginRecord("intStudentID")
        Session.Contents.Item("StrUserType") = LoginRecord("strUserType")
        Session.Contents.Item("CurPriceOrCost") = LoginRecord("curReservationOrCost")
        Session.Contents.Item("CurUtilityOrProfit") = LoginRecord("curUtilityOrProfit")
        Session.Contents.Item("BlnPlayStatus") = LoginRecord("BlnPlayStatus")
    Dim StrCheckMarketQuery
    StrCheckMarketQuery = "Select curOffer from MarketDB where intOfferer = " & Session.Contents.Item("intUserID")
    LoginRecord.Close
    LoginRecord.open StrCheckMarketQuery, DBUser, 2, 3, 1
    if not LoginRecord.EOF then
        Session.Contents.Item("CurOfferOrBid") = LoginRecord("curOffer")
    end if
    Else initialize their data, depending on current market conditions.
End If

'First, determine if they're a buyer or a seller, given the number of each in the current market
If Application.Contents.Item("intSellers") > Application.Contents.Item("intBuyers") then
    Session.Contents.Item("IntUserID") = LoginRecord("intStudentID")
    Session.Contents.Item("StrUserType") = "Buyer"
else
    Application.Contents.Item("intSellers") = (Application.Contents.Item("intSellers") + 1)
    Session.Contents.Item("IntUserID") = LoginRecord("intStudentID")
    Session.Contents.Item("StrUserType") = "Seller"
End If

' fetch supply and demand data from the scheduleDB
Dim scheduleRecord
Dim strSchedule
strSchedule = "SELECT * FROM ScheduleDB"
set scheduleRecord = server.CreateObject("ADODB.recordset")
set scheduleRecord = DBUser.Execute(strSchedule, , adcmdtext)

' now we initialize the user based on the market schedule settings stored in Schedule DB
Dim looper
looper = 0
' if the user is a buyer, use the demand curve
if Session.Contents.Item("strUserType") = "Buyer" then
Dim ReservationPrice
' start by initializing the reservation price to the minimum value
ReservationPrice = scheduleRecord.Fields("curDMin")
' so long as we haven't looped through once for each buyer, do the following
While looper <> Application.Contents.Item("intBuyers")
' increment the reservation price
ReservationPrice = ReservationPrice + scheduleRecord.Fields("curDInc")
' then reset it to the minimum if we've gone over the max value
If ReservationPrice >= scheduleRecord.Fields("curDMax") then
ReservationPrice = scheduleRecord.Fields("curDMin")
End If
' increment the looper
looper = looper + 1
Wend
Session.Contents.Item("CurPriceOrCosc") = ReservationPrice
Session.Contents.Item("BlnPlayStatus") = True
' else initialize them off of the supply curve
Else
Dim ProductionCost
' set production cost to the minimum value
ProductionCost = scheduleRecord.Fields("curSMin")
' loop through once for each seller
While looper <> Application.Contents.Item("intSellers")
' increment production cost
ProductionCost = ProductionCost + scheduleRecord.Fields("curSInc")
' reset to minimum if over maximum
If ProductionCost >= scheduleRecord.Fields("curSMax") then
ProductionCost = scheduleRecord.Fields("curSMin")
End If
' increment the looper
looper = looper + 1
Wend
' assign values to session object
Session.Contents.Item("CurPriceOrCosc") = ProductionCost
Session.Contents.Item("BlnPlayStatus") = True
End If

' write all changes back to the database
LoginRecord("strUserType") = Session.Contents.Item("strUserType")
LoginRecord("CurReservationOrCost") = Session.Contents.Item("CurPriceOrCosc")
LoginRecord.Update

End If

' Free up system resources
DBUser.Close
Set DBUser = Nothing
Set LoginRecord = Nothing
Set scheduleRecord = Nothing

Session.Contents.Item("StrResponse") = ""
Response.Redirect "MarketDisplay.asp"

End if
Market Simulation Display Subsystem

- MarketDisplay.asp
  - Top Market.asp
- CurrentBuyers.asp
- CurrentSellers.asp
- UserInfo.asp
- Logout.asp
- Graph.asp

Login Subsystem
- MakeSale.asp
- PostOffer.asp

Admin Subsystem
- PriceSeries.asp

User DB

Market DB

Schedule DB
<% Language=VBScript %>
<% Response.CacheControl = "Private" %>
<html>
<head>
title=Current Buyers</title>
<base target="_top">
</head>
<body bgcolor="#000000">
</head bgcolor="#000000">
</div align="center">
<center>
<table border="5" width="100%" cellpadding="3" cellspacing="5" bordercolorlight="#007F00" bordercolordark="#000000" bgcolor="#FFFFCC">
<tr>
<td colspan="3" bgcolor="#007F00">
<p align="center"><b><font color="#FFFFCC">Current Buyers</font></b></td>
</tr>
<tr>
<td bgcolor="#000000" align="center"><font color="#FFFFCC"><b>Buyer ID</b></font></td>
<td bgcolor="#000000" align="center"><font color="#FFFFCC"><b>Listed Bid</b></font></td>
<td colspan="3" width=62 bgcolor="#000000" align="center"><font color="#FFFFCC"><b>Accept?</b></font></td>
</tr>
</table>

First, we set up a holder recordset for the application-level market data, along with other essential local variables

Set Connection = server.CreateObject("ADODB.Connection")
Connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source= " & server.MapPath("fpdb/UserDB.mdb")
Connection.Open

Dim MarketRecord
Dim MarketQuery

Set MarketQuery = Server.CreateObject("ADODB.RecordSet")
MarketQuery = "Select a.intOfferer, a.curOffer, b.strUserType From UserDB as b, MarketDB as a Where b.intStudentID = a.intOfferer AND a.intAccepter = 0"
Set MarketRecord = connection.Execute(MarketQuery, ,adCmdText)

if not MarketRecord.EOF then
MarketRecord.MoveFirst
Do
if MarketRecord.Fields("strUserType") = "Buyer" then
if session.Contents.Item("IntUserID") = MarketRecord.Fields("intOfferer") then%
<tr>
<td><b><font color="#33CCFF">% MarketRecord.Fields("intOfferer") %</font></b></td>
</tr>
</center>
</table>

% end if %
</tr>
</table>

if session.Contents.Item("strUserType") = "Seller" and session.Contents.Item("blnPlayStatus") = true then%
<tr>
<td width=62><form method="POST" action="makeSale.asp" id="% Form & MarketRecord.Fields("intOfferer") %" name="% Form & MarketRecord.Fields("intOfferer") %"
<input type="hidden" name="BuyerID" value="% Marketrecord.Fields("intOfferer") %">
<input type="hidden" name="BuyerOffer" value="% Marketrecord.Fields("curOffer") %">
<input type="Submit" name="Accept" value="Accept">
</input>
</form>
</td>
</tr>
</table>

end if
loop until MarketRecord.EOF
</tr>
</table>
</center>
</div>
</body>
</html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <meta http-equiv="Cache-Control" content="Private">
  </head>
  <body>
    <table border="5" cellspacing="5" width="100%" bordercolorlight="#007500" bordercolordark="#000000" align="center">
      <thead class="center" bgcolor="#007500">
        <tr>
          <td bgcolor="#007500" colspan="3" align="center"><b><font color="#FFFFCC">Current Sellers</font></b></td>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td bgcolor="#000000" align="center" font color="#FFFFCC">Seller ID</td>
          <td bgcolor="#000000" align="center" font color="#FFFFCC">Listed Offer Price</td>
          <td align="center" font color="#FFFFCC">Accept?</td>
        </tr>
        <tr>
          <td bgcolor="#000000" align="center" font color="#FFFFCC">Seller ID</td>
          <td bgcolor="#000000" align="center" font color="#FFFFCC">Listed Offer Price</td>
          <td align="center" font color="#FFFFCC">Accept?</td>
        </tr>
        <tr>
          <td bgcolor="#000000" align="center" font color="#FFFFCC">Seller ID</td>
          <td bgcolor="#000000" align="center" font color="#FFFFCC">Listed Offer Price</td>
          <td align="center" font color="#FFFFCC">Accept?</td>
        </tr>
      </tbody>
    </table>
  </body>
</html>
Dim BuyerUtility
Dim SellerProfit
Dim IntNewTrans
Dim LngRecordsEffected

session.Contents.Item("strResponse") = ""

'Set up the database objects for database manipulation
Set Connection = server.CreateObject("ADODB.Connection")
Connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source= " & Server.MapPath("fpdb/UserDB.mdb")
Connection.Open
Set saleRecord = server.createobject("ADODB.RecordSet")

'First, check to make sure the person hasn't already made a trade
strStatusCheck = "Select blnPlayStatus, from UserDB where " & session.Contents.Item("intUserID") & " intStudentID"
session.Contents.Item("curUtilityOrProfit") = saleRecord.Fields("CurUtilityOrProfit")
session.Contents.Item("curOfferOrBid") = 0
session.Contents.Item("strResponse") = "We're sorry, but you have already completed the maximum number of transactions allowed in the Market."
Response.Redirect("MarketDisplay.asp")
end if

'second, check and see if the person is an administrator
if session.Contents.Item("strUserType") = "Admin" then
    session.Contents.Item("strResponse") = "We're sorry, but administrators cannot conduct transactions in the market."
    Response.Redirect("MarketDisplay.asp")
end if

'third, check if offer accepted has been accepted already from another existing login on a different machine
if session.Contents.Item("strUserType") = "Seller" then
    strCheckQuery = "SELECT * FROM MarketDB WHERE (" & Request.Form("BuyerID") & " = intOfferer OR " & Request.Form("SellerID") & " = intAccepter) and intTransaction <> 0"
else
    strCheckQuery = "SELECT * FROM MarketDB WHERE (" & Request.Form("BuyerID") & " = intOfferer OR " & Request.Form("SellerID") & " = intAccepter) and intTransaction <> 0"
end if

set saleRecord = connection.Execute(strCheckQuery, , adCmdText)
if not saleRecord.EOF then
    if session.Contents.Item("strUserType") = "Seller" then
        session.Contents.Item("strResponse") = "We're sorry, but your transaction cannot be completed. The bid has already been accepted by another party"
    else
        session.Contents.Item("strResponse") = "We're sorry, but your transaction cannot be completed. The offer has already been accepted by another party"
    end if
    Response.Redirect("MarketDisplay.asp")
else
    'if neither of these is the case, then proceed with the transaction
    first, we perform the transaction for sellers
    if session.Contents.Item("strUserType") = "Seller" then
        'Create Query Strings
        strGetTrans = "SELECT (max (intTransaction) + 1) as intNewTrans from MarketDB"
        strDeleteQuery = "Delete from MarketDB where intOfferer = " & Request.Form("BuyerID")
        strGetAcceptedOffer = "Select curReservationOrCost from UserDB where intStudentID = " & Request.Form("BuyerID")
        strUpdateMarket = "Update MarketDB Set intAccepter = " & Request.Form("BuyerID") & ", curOffer = " & Request.Form("BuyerOffer") & ", intTransaction = " & IntNewTrans & " where intOfferer = " & session.Contents.Item("intUserID")
        call connection.Execute(strUpdateMarket, LngRecordsEffected, adCmdText)
        if no record existed to update with the previous query then we need to insert a new row
        if LngRecordsEffected = 0 then
            'first we set up the insertion string
            strInsertToMarket = "INSERT into MarketDB (intOfferer, intAccepter, curOffer, intTransaction) VALUES (" & Request.Form("BuyerID") & ", " & Request.Form("BuyerOffer") & ", " & IntNewTrans & ")"
            then we pop in the new row
            call connection.Execute(strInsertToMarket, , adCmdText)
            end if
        else
            'delete record in MarketDB (delete buyer record)
            call connection.Execute(strDeleteQuery, , adCmdText)
            'determine profit/utility for both parties
            set saleRecord = connection.Execute(strGetAcceptedOffer, ,adCmdText)
        end if
BuyerUtility = SaleRecord.fields("curReservationOrCost") - Request.form("BuyerOffer")
SellerProfit = Request.form("BuyerOffer") - Session.Contents.Item("CurPriceOrCost")

'set up query string dependant on above declared variables
strUpdateBuyer = "Update UserDB set intPlayStatus = 0, curUtilityOrProfit = " & buyerutility &" where intStudentID = " & Request.Form("BuyerID")
strUpdateSeller = "Update UserDB set intPlayStatus = 0, curUtilityOrProfit = " & sellerprofit &" where intStudentID = " & session.Contents.Item("intUserID")
'update new numbers to userDB, switch play status to false
call connection.Execute(strUpdateBuyer, , adCmdText)
call connection.Execute(strUpdateSeller, , adCmdText)

'update the session variables
session.Contents.Item("curUtilityOrProfit") = SellerProfit
session.Contents.Item("CurOfferOrBid") = 0
session.Contents.Item("blnPlayStatus") = false
'clean up memory and redirect
Connection.Close
set connection = nothing
set saleRecord = nothing
session.Contents.Item("strResponse") = "sale made"
Response.Redirect("MarketDisplay.asp")
else
'then we perform almost the same transaction for buyers
'Create Query Strings
strGetTrans = "Select (max(intTransaction) + 1) as intNewTrans from MarketDB"
strDeleteQuery = "Delete from MarketDB where intOfferer = " & session.Contents.Item("intUserIO")
strGetAcceptedOffer = "Select curReservationOrCost from UserDB where intStudentID = " & Request.Form("SellerID")
strUpdateMarket = "Update MarketOB Set intAccepter = " & session.Contents.Item("intUserIO") &", intTransaction = " & intNewTrans &" where intOfferer = " & Request.Form("SellerID")
set saleRecord = connection.Execute(strGetTrans , , adCmdText)
set intNewTrans = saleRecord.Fields("intNewTrans")
set up query string dependant on intNewTrans
call query to insert new record to database
intAccepter = " & session.Contents.Item("intUserIO") &",
intTransaction = " & intNewTrans &" where intOfferer = " & Request.Form("SellerID")
call connection.Execute(strUpdateMarket, , adCmdText)
'delete record in MarketDB (delete buyer record)
call connection.Execute(strDeleteQuery, , adCmdText)
'determine profit/utility for both parties
set BuyerUtility = connection.Execute(strGetAcceptedOffer, , adCmdText)
set SellerProfit = Request.Form("sellerOffer") - saleRecord.fields("curReservationOrCost")

'set up query string dependant on above declared variables
strUpdateBuyer = "Update UserDB set intPlayStatus = 0, curUtilityOrProfit = " & buyerutility &" where intStudentID = " & session.Contents.Item("intUserID")
strUpdateSeller = "Update UserDB set intPlayStatus = 0, curUtilityOrProfit = " & sellerprofit &" where intStudentID = " & Request.Form("SellerID")
'update new numbers to userDB, switch play status to false
call connection.Execute(strUpdateBuyer, , adCmdText)
call connection.Execute(strUpdateSeller, , adCmdText)

'update the session variables
session.Contents.Item("curUtilityOrProfit") = BuyerUtility
session.Contents.Item("CurOfferOrBid") = 0
session.Contents.Item("blnPlayStatus") = false
'clean up memory and redirect
Connection.Close
set connection = nothing
set saleRecord = nothing
session.Contents.Item("strResponse") = "sale made"
Response.Redirect("MarketDisplay.asp")
end if
<html>
<head>
<title>Market Display</title>
</head>

<frameset rows="70,*,215">
    <frame name="Greeting" scrolling="no" noresize target="_top" src="TopMarket.asp">
        <frame name="Buyers" scrolling="auto" target="_top" src="CurrentBuyers.asp">
            <frame name="Sellers" scrolling="auto" target="_top" src="CurrentSellers.asp">
                <frame name="Info" scrolling="auto" target="_top" src="UserInfo.asp">
                    <noframes>
                        <p>This page uses frames, but your browser doesn't support them.</p>
                    </noframes>
                </frame>
            </frame>
        </frame>
    </frame>
</frameset>

</body>
</html>
Dim Connection
Dim PostRecord
Dim strPostQuery
Dim strMarketDetails
Dim MarketRecord

'right off the bat, we clear the response string of previous responses, to avoid confusing repetition of old messages
session.Contents.Item("strResponse") = Request.ServerVariables("Path_Info")

'then we set up a bunch of database interaction objects
Set Connection = server.CreateObject("ADODB.Connection")
Connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source= " & Server.MapPath("fpdb/UserDB.mdb")
Connection.Open
set PostRecord = server.CreateObject('ADODB.recordset')
set MarketRecord = server.CreateObject('ADODB.recordset')

'we set up the string to retrieve current market conditions
strMarketDetails = "SELECT * FROM ScheduleDB"
set MarketRecord = connection.Execute(strMarketDetails, , adcmdText)

'check if the person is an administrator. If they are, they can't post
if session.Contents.Item("strUserType") = "Admin" then
  session.Contents.Item("strResponse") = "We're sorry, but administrators cannot conduct transactions in the market"
  Response.Redirect("MarketDisplay.asp")
end if

'check to make sure the input is valid. If not, send an error message
if !isNumeric(Request.Form("offer")) then
  session.Contents.Item("strResponse") = "Please enter a valid numeric value. No letters or non-numeric characters."
  end if

'here we clean up the memory allocated to the database objects
set PostRecord = nothing
set Connection = nothing
set MarketRecord = nothing
Response.Redirect("MarketDisplay.asp")
<HTML>
<HEAD>
<title>Time Series of Recent Transaction Clearing Prices</title>
</HEAD>
<BODY>
<p align="center"><u><font size="5">List of Completed Transactions</font></u></p>
<div align="center">
<table border="1" cellspacing="5" width="100" bordercolor="#000000">
<tr>
<td>Seller</td>
<td>Buyer</td>
<td>Transaction Price</td>
</tr>
<% Set Connection = server.CreateObject("ADODB.Connection")
Connection.ConnectionString = "Provider:Microsoft.Jet.OLEDB.4.0; Data Source= " & Server.MapPath("fpdb/UserDB.mdb")
Connection.Open
Dim PriceRecord
Dim StrPriceQuery

'Initialize Database objects
Set DataRecord = Server.createObject("ADODB.RecordSet")

'Initialize Query strings
StrDataQuery = "SELECT * FROM MarketDB WHERE intTransaction > 0 ORDER BY intTransaction"

'Retrieve data
Set DataRecord = Connection.Execute(StrDataQuery, ,adCmdText)

'Set up the DataRecord recordset with additional market data
if not DataRecord.EOF then
    DataRecord.MoveFirst
    'loop through each record, displaying along the way
    Do
        <tr>
            <td><%= DataRecord.Fields("intOfferer")%></td>
            <td><%= DataRecord.Fields("intAccepter")%></td>
            <td><%= DataRecord.Fields("curOffer")%></td>
        </tr>
    DataRecord.MoveNext
    Loop until DataRecord.EOF
end if
%
</table>
</BODY>
</HTML>
Double Auction Market Simulation

Welcome, <%= session.Contents.Item("strUserType") %>. You are a <%= session.Contents.Item("strUserType") %>' in the current market.
<t>
<title>Current User and Market Information</title>
<html>
<head>
<title>Current User and Market Information</title>
</head>
<body background="indexbkgd.jpg" link="#008000" vlink="#000080" alink="#C0C0C0">

<p align="center" style="line-height: 100%"><font size="3" color="#007000">Session Contents.Item("strUserType")'s Information</font></p>

<p align="center"><b><font size="3" color="#007000">Session Contents.Item("strResponse")</font></b></p>

@if session.Contents.Item("blnPlayStatus") = True and session.Contents.Item("strUserType") <> "Admin" then
<center>
</center>
</if>

<table border="5" cellspacing="4" bgcolor="#007000" bordercolorlight="#FFFFCC" bordercolordark="#000000" height="48">
<tr>
<td width="25%" height="36"><font color="#FFFFCC"><b>Current Bid:</b></font></td>
<td>
</td>
<td width="25%" height="36"><font color="#FFFFCC"><b>Current Offer Price:</b></font></td>
<td>
</td>
<td width="10%" height="36"><font color="#FFFFCC">formatCurrency(session.Contents.Item("CurOfferOrBid"), 2, vbTrue, vbFalse, vbTrue)</font></td>
</tr>
<tr>
<td width="10%" rowspan="3" height="40">
</td>
<td width="25%" height="12"><font color="#FFFFCC"><b>Status of Bid:</b></font></td>
<td width="25%" height="12"><font color="#FFFFCC"><b>Status of Offer:</b></font></td>
<td width="10%" height="12">Accepted</td>
<td width="10%" height="12">Standing</td>
<td width="10%" height="12">None Posted</td>
</tr>
<tr>
<td width="25%" height="36"><font color="#FFFFCC">formatCurrency(session.Contents.Item("CurPriceOrCost"), 2, vbTrue, vbFalse, vbTrue)</font></td>
</tr>
<tr>
<td width="10%" height="36"><font color="#FFFFCC">Reservation Price:</font></td>
<td>
</td>
<td width="25%" height="36"><font color="#FFFFCC">Production Cost:</font></td>
<td>
</td>
</tr>
</table>

<table border="3" width="100%" cellspacing="4" bgcolor="#007500" bordercolorlight="#FFFFFF" bordercolordark="#000000" height="48">
<tr>
<td width="25%" height="36"><b>Change or Post Bid:</b></td>
<td>
<form method="POST" name="offerform" action="PostOffer.asp">
<table border="0" cellpadding="0" bordercolorlight="#007000" bordercolordark="#000000" bgcolor="#007500">
<tr>
<td align="center">
<p style="line-height: 100%"><input name="offer" type="text" size="15"></p>
</td>
<td>
<button type="submit" name="submitoffer">OK</button>
</td>
</tr>
</table>
</form>
</td>
</tr>
<tr>
<td width="10%" height="36"><b>Reservation Price:</b></td>
<td>
</td>
<td width="25%" height="36"><font color="#FFFFCC">formatCurrency(session.Contents.Item("CurPriceOrCost"), 2, vbTrue, vbFalse, vbTrue)</font></td>
</tr>
</table>

</if>
</if>
</center>

</table>
</body>
</html>
```html
<!-- Code Block for generating current market winnings -->

```<

```html
<tr align="left"><b>Currency Price Floor</b></tr>
</td>
</tr>
<tr align="left"><b>Current Price Ceiling</b></tr>
</td>
</tr>

```
<html>
<head>
<title>Administrator Options</title>
</head>
<body background="indexbkgd.jpg">

<p align="center">Administrator Options</p>
<table border="10" cellspacing="5" width="100%" bordercolorlight="#008000" bordercolordark="#000000" bgcolor="#FFFFCC" height="304">
<tr>
<td width="100%" colspan="2" height="55">This button will take you to the current market simulation page. The current players as well as their bids and offers will be displayed.</td>
</tr>
<tr>
<td width="41%" height="105">
<applet code="fphover.class" codebase="." width="238" height="40" VIEWASTEXT>
  <param name="color" value="#008000">
  <param name="textcolor" value="#FFFFCC">
  <param name="url" valuetype="ref" value="MarketDisplay.asp">
  <param name="effect" value="bevelOut">
  <param name="hovercolor" value="#000000">
  <param name="font" value="TimesRoman">
  <param name="fontsize" value="18">
  <param name="fontstyle" value="regular">
  <param name="text" value="Go to Market Simulation">
</applet>
</td>
<td width="59%" height="105">This button will pull up a web page that contains all of the current market summary information. Each player and their specific market information will be given in a printable format.</td>
</tr>
<tr>
<td width="41%" height="105">
<applet code="fphover.class" codebase="." width="238" height="40" VIEWASTEXT>
  <param name="color" value="#008000">
  <param name="textcolor" value="#FFFFCC">
  <param name="url" valuetype="ref" value="Current Market Info.asp">
  <param name="effect" value="bevelOut">
  <param name="hovercolor" value="#000000">
  <param name="font" value="TimesRoman">
  <param name="fontsize" value="18">
  <param name="fontstyle" value="regular">
  <param name="text" value="Display Current Market Data">
</applet>
</td>
<td width="59%" height="105">This button will reset the current market. All buyer and seller ID numbers will remain, however, all bids and offers will be RESET to ZERO, and the supply and demand curves will be restored to their default functions.</td>
</tr>
<tr>
<td width="41%" height="105">
<applet code="fphover.class" codebase="." width="238" height="40" VIEWASTEXT>
  <param name="color" value="#008000">
  <param name="textcolor" value="#FFFFCC">
  <param name="url" valuetype="ref" value="Market Schedules.asp">
  <param name="effect" value="bevelOut">
  <param name="hovercolor" value="#000000">
  <param name="font" value="TimesRoman">
  <param name="fontsize" value="18">
  <param name="fontstyle" value="regular">
  <param name="text" value="Display Market Schedules">
</applet>
</td>
<td width="59%" height="105">This button will take you to the current market simulation page. The current players as well as their bids and offers will be displayed.</td>
</tr>
</table>
</body></html>
<table>
  <tr>
    <td width="59%" height="105">Click here to view the current market demand and supply schedules</td>
  </tr>
  <tr>
    <td width="41%" height="105">
      <p align="center">
        <applet code="fphover.class" codebase="./" width="238" height="40" VIEWASTEXT>
          <param name="color" value="#008000">
          <param name="textcolor" value="#FFFFFF">
          <param name="hovercolor" value="#000000">
          <param name="font" value="TimesRoman">
          <param name="fontsize" value="18">
          <param name="fontstyle" value="regular">
          <param name="text" value="Change Market Parameters">
          <param name="url" value="/Demand and Supply.asp">
        </applet>
      </p>
    </td>
  </tr>
  <tr>
    <td width="59%" height="105">Click here to alter the characteristics of the demand and supply curves, and set other market parameters</td>
  </tr>
  <tr>
    <td width="41%" height="105">
      <p align="center">
        <applet code="fphover.class" codebase="./" width="238" height="40" VIEWASTEXT>
          <param name="color" value="#008000">
          <param name="textcolor" value="#FFFFFF">
          <param name="hovercolor" value="#000000">
          <param name="font" value="TimesRoman">
          <param name="fontsize" value="18">
          <param name="fontstyle" value="regular">
          <param name="text" value="Logout">
          <param name="url" value="/Logout.asp">
        </applet>
      </p>
    </td>
  </tr>
  <tr>
    <td width="59%" height="105">Logout Administrator</td>
  </tr>
</table>
<script language="JavaScript" runat="Client">
var msg = "Are you certain you wish to erase all market data?";
if(confirm(msg))
    location.replace("clearMarket.asp");
else
    location.replace("Admin.asp");
</script>
'for adjustment later
Set Connection = server.CreateObject("ADODB.Connection")
Connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source= " & Server.MapPath("fpdb/UserDB.mdb")
Response.write "Provider=Microsoft.Jet.OLEDB.4.0; Data Source= " & Server.MapPath("fpdb/UserDB.mdb")
Connection.Open
'for adjustment later
dim strClearQuery
dim strRestoreQuery

'start with the query to clear all market data
strClearQuery = "DELETE FROM MarketDB WHERE intOfferer <> 0 OR intOfferer = 0"
strRestoreQuery = "UPDATE UserDB SET strUserType = '', curReservationOrCost = 0, curUtilityOrProfit = 0, curPlayStatus = True WHERE strUser <> 'Admin"
strScheduleClear = "DELETE FROM ScheduleDB"
strScheduleQuery = "INSERT INTO ScheduleDB (curDMax, curDMin, curDInc, curSMax, curSMin, curSInc, curPFloor, curPCeiling, timGameEnd, timGameStart) Values (14, 2, 1, 14, 2, 1, -10000, -10000, '12/31/9999', '12/31/9999')"

call Connection.Execute(strClearQuery, _adCmdText)
call Connection.Execute(strRestoreQuery, _adCmdText)
call Connection.Execute(strScheduleClear, _adCmdText)
call Connection.Execute(strScheduleQuery, _adCmdText)

Application.Contents.Item("intSellers") = 0
Application.Contents.Item("intBuyers") = 0
Application.Contents.Item("intTotalSales") = 0
Application.Contents.Item("curAvgPrice") = 0

set connection = nothing
Response.Redirect("admin.asp")
<% Response.CacheControl = "Private" %>
<html>
<head>
<title>Current Market Data</title>
<base target="_blank">
</head>
<body>
<p align="center"><u><font size="S">Current Market Data</font></u></p>
<div align="center">
<br />
<table border="1" cellspacing="5" width="100%" bordercolordark="#000000">
<tr>
<td>ID Number</td>
<td>Name</td>
<td>Buyer/Seller</td>
<td>Resv. Price/cost</td>
<td>Bid/Offer</td>
<td>Status of Offer</td>
<td>Surplus/Profit</td>
</tr>
<tr>
<td><% Set DataRecord = Server.CreateObject("ADODB.RecordSet") Set DataRecord = Server.CreateObject("ADODB.RecordSet") %></td>
<td><% Set DataRecord = Server.CreateObject("ADODB.RecordSet") Set DataRecord = Server.CreateObject("ADODB.RecordSet") %></td>
<td><% Set DataRecord = Server.CreateObject("ADODB.RecordSet") Set DataRecord = Server.CreateObject("ADODB.RecordSet") %></td>
<td><% Set DataRecord = Server.CreateObject("ADODB.RecordSet") Set DataRecord = Server.CreateObject("ADODB.RecordSet") %></td>
<td><% Set DataRecord = Server.CreateObject("ADODB.RecordSet") Set DataRecord = Server.CreateObject("ADODB.RecordSet") %></td>
<td><% Set DataRecord = Server.CreateObject("ADODB.RecordSet") Set DataRecord = Server.CreateObject("ADODB.RecordSet") %></td>
<td><% Set DataRecord = Server.CreateObject("ADODB.RecordSet") Set DataRecord = Server.CreateObject("ADODB.RecordSet") %></td>
<tr>
</tr>
</table>
</div>
</body>
</html>
<html>
<title>Current Market Schedules</title>
<body>
<table border="1" cellspacing="5" width="100%" bordercolordark="#000000">
<tr>
<td width="100"><p align="center"><font size="5">Current Market Schedules</font></p></td>
</tr>
</table>
</body>
</html>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Current Market Schedules</title>
</head>
<body>
<table border="1" cellspacing="5" width="100%" bordercolordark="#000000">
<tr>
<td width="50"><p align="center">Demand</p></td>
<td width="50"><p align="center">Supply</p></td>
</tr>
</table>
<table border="1" cellspacing="5" width="100%" bordercolordark="#000000">
<tr>
<td width="25"><p align="center">Number of Buyers</p></td>
<td width="25"><p align="center">Price</p></td>
<td width="25"><p align="center">Number of Sellers</p></td>
<td width="25"><p align="center">Price</p></td>
</tr>
</table>

'first, set up the database connection
Dim connection
Dim marketData
Dim holder
Dim looper
Dim runningCount
runningCount = 0
looper = 0
set connection = server.CreateObject("ADODB.connection")
set marketData = server.createobject("ADODB.Recordset")
set holder = server.createobject("ADODB.Recordset")
connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source= " & Server.MapPath("fpdb/UserDB.mdb")
connection.open
' then we fetch the basic market data with this connection
strMarketData = "SELECT * FROM ScheduleDB"
set marketData = connection.execute(strMarketData, , adcmdText)
strPlayers = "SELECT * FROM UserDB"
set holder = connection.execute(strPlayers, , adcmdText)

' first, we count up the entries in the demand curve
runningCount = 0
holder.MoveFirst
while not holder.EOF
   if holder.Fields("curReservationOrCost") >= (marketData.Fields("curDMin") + looper*marketData.Fields("curDInc")) and holder.Fields("strUserType") = "Buyer" then
      runningCount = runningCount + 1
   end if
   holder.MoveNext
wend

' then, we count up the entries in the supply curve
runningCount = 0
holder.MoveFirst
while not holder.EOF
   if holder.Fields("curReservationOrCost") <= (marketData.Fields("curSMin") + looper*marketData.Fields("curSInc"))
      runningCount = runningCount + 1
   end if
   holder.MoveNext
wend
</body>
</html>
end if
holder.movenext
wend

<% looper = looper + 1 %>
</tr>
</table>
</BODY>
</HTML>
Market Parameter Control Subsystem

Demand and Supply.asp

Admin Subsystem

ChangeSchedule.asp
- Set Market Restrictions
- Set Supply and Demand Curves
- Shift Supply and Demand Curves
- Set Simulation Time Restrictions

Schedule DB
<%@ Language=VBScript %>
<
session.Contents.Item("strResponse") = "*
' set up the database connection
' and initialize market data
************************************************************************************

dim connection
dim holder
set connection = server.CreateObject("ADODB.connection")
set holder = server.createobject("ADODB.Recordset")
connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=" & Server.MapPath("fpdb/UserDB.mdb")
connection.open

' first, fetch some check data from the database
strDataCheck = "SELECT * FROM ScheduleDB"
set holder = connection.Execute(strDataCheck, , adcmdtext)

' if there exist no current market schedule, abort
if holder.EOF then
call endModule(connection, holder, "The Market has not yet been initialized. No changes were made")
end if

' **********************************************************************************

'Main Redirection Statement

' **********************************************************************************

' if the user sent data for simple demand and supply curve shifts, do the following
if Request.Form("Admin Option") = "SimpleSD" then
call SimpleSandD(connection, holder, Request.form("SShift"), Request.form("DShift"))
endif

' if the user sent data for time control options, we execute this code
elseif Request.Form("Admin Option") = "TimeControl" then
call setTimes(connection, holder, Request.Form("Begin"), Request.Form("End"), holder.Fields("timGameStart"), holder.Fields("timGameEnd"))
endif

' if the user set a price floor or ceiling, do the following
elseif Request.Form("Admin Option") = "MarketRestrictions" then
call MarketRestriction(connection, holder, Request.Form("floor"), holder.Fields("curPFloor"))
endif

' if the form sent was for advanced supply and demand curve options, do the following
elseif Request.Form("Admin Option") = "AdvancedSD" then
call AdvanceSandD(connection, holder, Request.form("DMin"), Request.form("DMax"), Request.form("DInc"), Request.form("DMin"), Request.form("DMax"), Request.form("DInc"))
endif

' **********************************************************************************

' End Main Redirection Statement

' **********************************************************************************

' after all interaction, clean up and exit
call endModule(connection, holder, "")

' **********************************************************************************

' Function Bodies of main task functions

' **********************************************************************************

sub MarketRestriction(connection, holder, nCeiling, nFloor, cCeiling, cFloor)

' if the price floor is greater than the price ceiling, we redirect here and report an error
'since we don't know exactly what data the may be
if (isnumeric(nFloor) and isnumeric(nCeiling)and (nFloor > nCeiling)) then
call endModule(connection, holder, "The price floor is greater than the price ceiling. Please re-submit data")
endif

' this section removed for debugging
elseif (isnumeric(nFloor) and (nFloor > cCeiling) and (cFloor <> -10000)) then
call endModule(connection, holder, "The price floor is greater than the price ceiling. Please re-submit data")
endif

' check for a price floor change
if the value was a valid numeric value, set the price floor
if isnumeric(nFloor) then
strFloorSet = "UPDATE ScheduleDB SET curPFloor = "+ nFloor
call connection.Execute(strFloorSet, , adcmdtext)
endif

' if it wasn't, check for a "remove" statement
elseif nFloor = "remove" or nFloor = "Remove" or nFloor = "REMOVE" then
strRestrictionClear = "UPDATE ScheduleDB SET curPFloor = -10000"
call connection.Execute(strRestrictionClear, , adcmdText)

' if neither of these is the case, report an error to the user
else
  if nFloor <> "" then
call endModule(connection, holder, "Invalid data. Please re-submit changes.")
end if
end if

' check for a price ceiling change
' if the value was a valid numeric value, set the price ceiling
if isnumeric(nCeiling) then
  strCeilingSet = "UPDATE ScheduleDB SET curPCeiling = " & nCeiling
  call connection.Execute(strCeilingSet, , adcmdText)
' if it wasn't, check for a "remove" statement
elseif nCeiling = "remove" or nCeiling = "Remove" or nCeiling = "REMOVE" then
  strRestrictionClear = "UPDATE ScheduleDB SET curPceiling = -10000"
call connection.Execute(strRestrictionClear, , adcmdText)
' if neither of these is the case, report an error to the user
else
  if nCeiling <> "" then
    call endModule(connection, holder, "Invalid data. Please re-submit changes.")
  end if
end if

end sub

' check for a price ceiling change
' if the value was a valid numeric value, set the price ceiling
if isnumeric(nCeiling) then
  strCeilingSet = "UPDATE ScheduleDB SET curPCeiling = " & nCeiling
  call connection.Execute(strCeilingSet, , adcmdText)
' if it wasn't, check for a "remove" statement
elseif nCeiling = "remove" or nCeiling = "Remove" or nCeiling = "REMOVE" then
  strRestrictionClear = "UPDATE ScheduleDB SET curPceiling = -10000"
call connection.Execute(strRestrictionClear, , adcmdText)
' if neither of these is the case, report an error to the user
else
  if nCeiling <> "" then
    call endModule(connection, holder, "Invalid data. Please re-submit changes.")
  end if
end if

end sub
'if neither, report an error
if not (isnumeric(SShift) or isnumeric(DShift)) then
    call endModule(connection, holder, "Invalid data. Please restrict entries to numbers.")
end if

end sub

sub setTimes(connection, holder, BeginTime, EndTime, cBeginTime, cEndTime)
'before we even start, if the user input values that make no sense, we send them straight back with an error
if (isDate(BeginTime) and isDate(EndTime)) then
    if (datediff(d, EndTime, BeginTime) <= 0) then
        call endModule(connection, holder, "The Begin Date is later than the End Date. Please re-submit data.")
    elseif (isDate(BeginTime) and (datediff(d, cEndTime, BeginTime) <= 0) and (cEndTime <> #12/31/9999#)) then
        call endModule(connection, holder, "The Begin Date is later than the End Date. Please re-submit data.")
    elseif (isDate(EndTime) and (datediff(d, EndTime, cBeginTime) <= 0) and (cBeginTime <> #12/31/9999#)) then
        call endModule(connection, holder, "The Begin Date is later than the End Date. Please re-submit data.")
    end if
end if

'check for a game begin date
if BeginTime <> "" then
    'ensure the data is valid
    if isDate(BeginTime) then
        strFloorSet = "UPDATE ScheduleDB SET timGameStart = "; BeginTime & ";"
        call connection.Execute(strFloorSet, , adcmdText)
    'if the data is not a date, check for a remove command
    elseif BeginTime = "remove" or BeginTime = "Remove" or BeginTime = "REMOVE" then
        strRestrictionClear = "UPDATE ScheduleDB SET timGameStart = ";'12/31/9999""
        call connection.Execute(strRestrictionClear, , adcmdText)
    'if we get to this 'else', the data is invalid, and we ignore it and return an error
    else
        call endModule(connection, holder, "Invalid data. Please re-submit changes.")
    end if
end if

'check for a game end date
if EndTime <> "" then
    'ensure the data is valid
    if isDate(EndTime) then
        strCeilingSet = "UPDATE ScheduleDB SET timGameEnd = "; EndTime & ";"
        call connection.Execute(strCeilingSet, , adcmdText)
    'if the data is not a date, check for a remove command
    elseif EndTime = "remove" or EndTime = "Remove" or EndTime = "REMOVE" then
        strRestrictionClear = "UPDATE ScheduleDB SET timGameEnd = ";'12/31/9999""
        call connection.Execute(strRestrictionClear, , adcmdText)
    'if we get to this 'else', the data is invalid, and we ignore it and return an error
    else
        call endModule(connection, holder, "Invalid data. Please re-submit changes.")
    end if
end if

sub endModule(connection, holder, message)
    session.Contents.Item("strResponse") = message
    connection.Close
    set connection = nothing
    set holder = nothing
    response.Redirect("Demand and Supply.asp")
end sub
'first, set up the database connection
dim connection
dim marketData
set connection = server.CreateObject("ADODB.connection")
set marketData = server.CreateObject("ADODB.Recordset")
connection.connectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source= " & server.MapPath("fpdb/UserDB.mdb")
connection.open
strMarketData = "SELECT * FROM ScheduleDB"
set marketdata = connection.Execute(strMarketData, adcmdText)
%
<table width = "100%" BORDER="5" CELLPADDING="5" bordercolorlight="#999900" bordercolordark="#666633">
<tr>
<td colspan = "2" width = "50%">
<table width = "100%" BORDER="5" CELLPADDING="5" bordercolorlight="#999900" bordercolordark="#666633">
<tr>
<td colspan = "2" width = "50%">
<table width = "100%" BORDER="5" CELLPADDING="5" bordercolorlight="#999900" bordercolordark="#666633">
<tr>
<td colspan = "4" width = "100%">
<table width = "100%" BORDER="5" CELLPADDING="5" bordercolorlight="#999900" bordercolordark="#666633">
<tr>
<td colspan = "2" width = "50%">
<table width = "100%" BORDER="5" CELLPADDING="5" bordercolorlight="#999900" bordercolordark="#666633">
<tr>
<td colspan = "2" width = "50%">
<table width = "100%" BORDER="5" CELLPADDING="5" bordercolorlight="#999900" bordercolordark="#666633">
<tr>
<td colspan = "2" width = "50%">
<table width = "100%" BORDER="5" CELLPADDING="5" bordercolorlight="#999900" bordercolordark="#666633">
<tr>
<td colspan = "2" width = "50%">
<table width = "100%" BORDER="5" CELLPADDING="5" bordercolorlight="#999900" bordercolordark="#666633">
<p align="center">Current Maximum Reservation Price</p>
</td>
</tr>
<tr>
<td>
<font>
<p align="center">Current Maximum Production Cost</p>
</font>
</td>
<td>
<font>
<p align="center">Current Production Cost Increment</p>
</font>
</td>
</tr>
<tr>
<td>
<font>
<p align="center">Current Price Floor</p>
</font>
</td>
<td>
<font>
<p align="center">Current Price Ceiling</p>
</font>
</td>
</tr>
<tr>
<td>
<font>
<p align="center">Game Start Date</p>
</font>
</td>
<td>
<font>
<p align="center">#12/31/9999#</p>
</font>
</td>
</tr>
<table width="100%" BORDER="5" CELLPADDING="5" bordercolorlight="#999900" bordercolordark="#666633">
  <form method="POST" name="MarketRestrictions" action="ChangeSchedule.asp">
    <input cype="hidden" name="Admin Option" value="MarketRestrictions"></input>
    <tr>
      <td colspan="4">
        <p align="center">Market Restriction Options portion of screen</p>
      </td>
    </tr>
    <tr>
      <td colspan="4">
        <input type="submit" value="Submit" name="B1"></input>
      </td>
    </tr>
  </form>
  <tr>
    <td width="50%">
      <font>
        This form allows the administrator to add market restrictions in the form of price floors or ceilings. To remove existing restrictions, type "remove" in the input box.</font>
    </td>
  </tr>
</table>

---

**Garne End Date**

- If `MarketData.Fields("timGameEnd")` <= #12/31/9999 then
- `MarketData.Fields("timGameEnd")`
- Else
- None

---

**Market Restriction Options**

- **Price Floor**
  - `NAME="floor" TYPE="text" SIZE="15"`

- **Price Ceiling**
  - `NAME="ceiling" TYPE="text" SIZE="15"`
Simple Supply and demand options portion of screen

Game Time Control options portion of screen
<form method="POST" name="scheduleFormAdvanced" action="ChangeSchedule.asp">
<table width="100%" border="S" cellpadding="S" bordercolorlight="#999900" bordercolordark="#666633">
<tr>
<td colspan="2" width="50%">
<p align="center"><b>Advanced Supply and Demand Options</b></p>
</td>
</tr>
<tr>
<td>
<p align="center"><b>Minimum Reservation Price</b></p>
</td>
<td width="125">
<input name="DMin" type="text" size="15"></td>
</tr>
<tr>
<td>
<p align="center"><b>Minimum Production Cost</b></p>
</td>
<td width="125">
<input name="SMin" type="text" size="15"></td>
</tr>
<tr>
<td>
<p align="center"><b>Maximum Reservation Price</b></p>
</td>
<td width="125">
<input name="Max" type="text" size="15"></td>
</tr>
</table>
</form>
<table>
<thead>
<tr>
<th></th>
<th>Maximum Production Cost</th>
<th>Production Cost Increment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DMax</strong></td>
<td><code>&lt;input NAME=&quot;DMax&quot; TYPE=&quot;text&quot; SIZE=&quot;15&quot; MAXLENGTH=&quot;132&quot;&gt;</code></td>
<td><code>&lt;input NAME=&quot;SInc&quot; TYPE=&quot;text&quot; SIZE=&quot;15&quot; MAXLENGTH=&quot;132&quot;&gt;</code></td>
</tr>
<tr>
<td><strong>SMax</strong></td>
<td><code>&lt;input NAME=&quot;SMax&quot; TYPE=&quot;text&quot; SIZE=&quot;15&quot; MAXLENGTH=&quot;132&quot;&gt;</code></td>
<td><code>&lt;input NAME=&quot;SInc&quot; TYPE=&quot;text&quot; SIZE=&quot;15&quot; MAXLENGTH=&quot;132&quot;&gt;</code></td>
</tr>
</tbody>
</table>

**Reservation Price Increment:**
```
<tr><td colspan = "4"> <input type="submit" value="Submit" name="Bl"> </td></tr>
</table>
</td>
</tr>
<td colspan = "2"> <a href="Admin.asp" target="_top">Return to administrator options page</a> </td>
</font>
</tr>
</table>
</center>
</BODY>
</HTML>
Global and Miscellaneous Code Modules
<SCRIPT LANGUAGE='VBScript' RUNAT='Server'>

FrontPage_StartSession "==FrontPage Generated==" 

Sub FrontPage_StartSession
On Error Resume Next
if Len(Application("FrontPage VRoot")) > 0 then Exit Sub
' discover the VRoot for the current page;
' walk back up VPath until we find global.asa
Vroot = Request.ServerVariables("PATH_INFO")
strGl = "global.asa"
strG2 = "Global.asa~"
idx = 0
while Len(Vroot) > 1
   if FrontPage_FileExists(Server.MapPath(Vroot & strGl)) then exit do
   if Right(Vroot, 1) = "~" then Vroot = Left(Vroot, Len(Vroot) - 1)
   iCount = iCount + 1
   if iCount > 100 then
      Vroot = "../
   end if
end while
end if

loop 
' map all URL= attributes in _ConnectionString variables
Application.Lock
if Len(Application("FrontPage VRoot")) = 0 then
   Application("FrontPage VRoot") = Vroot
   UrlVarArray = Application("FrontPage UrlVars")
   for i = 0 to UBound(UrlVarArray)
      FrontPage_MapUrl(UrlVarArray(i))
   next
end if
Application.Unlock
End Sub

Function Frontpage_FileExists(fspath)
On Error Resume Next
FrontPage_FileExists = False
set fs = CreateObject("Scripting.FileSystemObject")
Err.Clear
set istream = fs.OpenTextFile(fspath)
if Err.Number = 0 then
   FrontPage_FileExists = True
   istream.Close
end if
set istream = Nothing
set fs = Nothing
End Function

Sub FrontPage_MapUrl(AppVarName)
' convert URL attribute in conn string to absolute file location
strVal = Application(AppVarName)
strKey = "URL:::
idxStart = InStr(strVal, strKey)
If idxStart = 0 Then Exit Sub
idxStart = idxStart + Len(strKey)
idxEnd = InStr(idxStart, strVal, ":")
If idxEnd = 0 Then
   strVal = Mid(strVal, idxStart)
Else
   strVal = Mid(strVal, idxStart, idxEnd - idxStart)
End If
strURL = Mid(strVal, idxStart)
End Sub

Sub Application_OnStart
'==FrontPage Generated - startspan==
Dim FrontPage_UrIVars(3)
'--Project Data Connection
Application("new-page_l_ConnectionString") = "DRIVER={Microsoft Access Driver (*.mdb)};DBQ=URL=fpdb/UserDB.mdb"
FrontPage_UrIVars(0) = "new-page_l_ConnectionString"
Application("new-page_l_ConnectionTimeout") = 15
Application("new-page_l_CommandTimeout") = 30
Application("new-page_l_CursorLocation") = 3
Application("new-page_l_RuntimeUserName") = ""
Application("new-page_l_RuntimePassword") = ""
</SCRIPT>
Application("FrontPage.UrlVars") = FrontPage.UrlVars

'This is code to initialize the values held globally by the application, most of them application
'scope market variable statistics
Dim intSellers
Dim intBuyers
Dim intRecordsEffected
Dim intTotalSales
Dim curAvgPrice
Dim DBInit
Dim Record
Dim TimGameStart
Dim TimGameEnd
Dim StrSellersQuery
Dim StrBuyersQuery
Dim StrTotalSalesQuery
Dim StrCurrentMarketQuery
Dim StrMarketConnectionString
Dim StrDatesQuery

'Here we put together the SQL Query String Por fetching User data
StrSellersQuery = "Select count (strUser) as intSellers From UserDB Where strUserType = 'Seller'"
StrBuyersQuery = "Select count (strUser) as intBuyers From UserDB Where strUserType = 'Buyer'"
StrTotalSalesQuery = "Select Count(intAccepter) as intTotalSales, AVG(curOffer) as curAvgPrice From MarketDB Where
intAccepter <> 0"
StrCurrentMarketQuery = "Select a.intOfferer, a.curOffer, b.strUserType From UserDB as b, MarketDB as a Where b.
intStudentID = a.intOfferer AND a.intAccepter = 0"
StrDatesQuery = "Select timGameStart, timGameEnd from ScheduleDB"

StrMarketConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source= " & Server.MapPath("fpdb/UserDB.mdb")

'This creates and sets the database connection for retrieving statistic information from the Database
Set DBInit = Server.CreateObject("ADODB.Connection")
DBInit.ConnectionString = StrMarketConnectionString
DBInit.Open

'and then we query the database using the SQL statement defined above to retrieve total sellers in market
Set Record = DBInit.Execute(StrSellersQuery, intRecordsEffected, adCmdText)
'if records were returned, then initialize application scope vars
if not Record.EOF then
  Record.movefirst
  intSellers = Record.Fields("intSellers")
End If

'and then we query the database using the SQL statement defined above to retrieve total buyers in market
Set Record = DBInit.Execute(StrBuyersQuery, intRecordsEffected, adCmdText)
'if records were returned, then initialize application scope vars
if not Record.EOF then
  Record.movefirst
  intBuyers = Record.Fields("intBuyers")
End If

'and then we query the database using the SQL statement defined above to retrieve total sales in market
Set Record = DBInit.Execute(StrTotalSalesQuery, intRecordsEffected, adCmdText)
'if records were returned, then initialize application scope vars
if not Record.EOF then
  Record.movefirst
  intTotalSales = Record.Fields("intTotalSales")
curAvgPrice = Record.Fields("curAvgPrice")
End If

'now we initialize the time restriction variables, if any time restrictions on the game exist
Set Record = DBInit.Execute(StrDatesQuery, intRecordsEffected, adCmdText)
if not Record.EOF then
  Record.movefirst
timGameStart = Record.Fields("timGameStart")
timGameEnd = Record.Fields("timGameEnd")
End If

'Clear variables to free up memory
Set Record = nothing
Set DBInit = nothing

End Sub

*************************************************************************
'Sub Session_OnStart
*************************************************************************

Sub Session_OnStart

'Declare session scope variables
'initialize session scope variables
Dim IntUserID

*************************************************************************
Dim StrUserType
Dim CurPriceOrCost
Dim CurUtilityOrProfit
Dim CurOfferOrBid
Dim BinPlayStatus
Dim StrResponse
Dim StrStartPage
Dim StrStart
Dim User
Dim Password

IntUser ID = 0
StrUserType = ""
CurPriceOrCost = 0
CurUtilityOrProfit = 0
CurOfferOrBid = 0
BinPlayStatus = False
StrResponse = ""
StrStartPage = Request.ServerVariables("PATH_INFO")
StrStart = "Login.asp"

'make sure the user is not trying to access some other page than the start up page
'if they are, redirect them to the start up page
If (StrComp(Right(StrStartPage, Len(StrStart)), StrStart, 1)) then
    Response.Redirect("Login.asp")
End If

End Sub

</SCRIPT>
<html>
<head>
<title>logout</title>
</head>
<body background="indexbkgd.jpg" link="#007500" vlink="#0000DD" alink="#808080">
<p align="center">Thanks for playing the Market Simulacion!</p>
<p align="center"><a href="login.asp" target="_top">Return to LOGIN</a></p>
</body>
</html>