1

# Design Discussion Space: Creating a 3-D Discussion Environment over the Internet

by

Danai Kuangparichat

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

June 1998

Signature of Author: ____________________
Department of Electrical Engineering and Computer Science
January 15, 1998

Certified by: [illegible]
Ronald MacNeil
~~Principal Research~~ Associate, Media Laboratory
Thesis Supervisor

Accepted by: ____________________
Frederic R. Morgenthaler
Chairman, Department Committee on Graduate Thesis

# Design Discussion Space: Creating a 3-D Discussion Environment over the Internet

by

Danai Kuangparichat

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

## ABSTRACT

Design Discussion Space (DDS) is an Internet application programmed in Java 1.0 and VRML 2.0 that supports three-dimensional discussion environments. DDS is one of the first tools that support discussions of three-dimensional objects on the Internet. After users log in, they can join any discussion room, or create a new discussion room with a topic of their choice. Once inside a discussion room, users can import two-dimensional images or three-dimensional objects, and post comments to contribute to the ongoing discussion. The content of discussions are preserved from session to session; so users can log out and come back later to check the progress of any discussion. The application also supports chatting, similar to zephyr-writing, as well as a history feature, which displays the activity in the discussion room since its creation.

Although DDS has not been fully implemented, its design has been completely laid out, and its implementation has reached a point where it is usable. A major obstacle arose when the VRML browser being used to develop DDS lost its Java functionality. Although beta versions of this browser had supported Java, the browser's first non-beta version did not. Because no other VRML browser that supported Java could be found, further testing could not be performed.

Currently, about ninety percent of the application has been coded and tested. Another seven to eight percent has been written, but can not yet be tested. As soon as a Java-capable VRML browser comes out, development of DDS should continue, based on the documentation presented herein.

Thesis Supervisor: Ronald MacNeil
Title: Principle Research Associate, Media Laboratory

# TABLE OF CONTENTS

# 1. INTRODUCTION

## 1A. The Civiscape Project

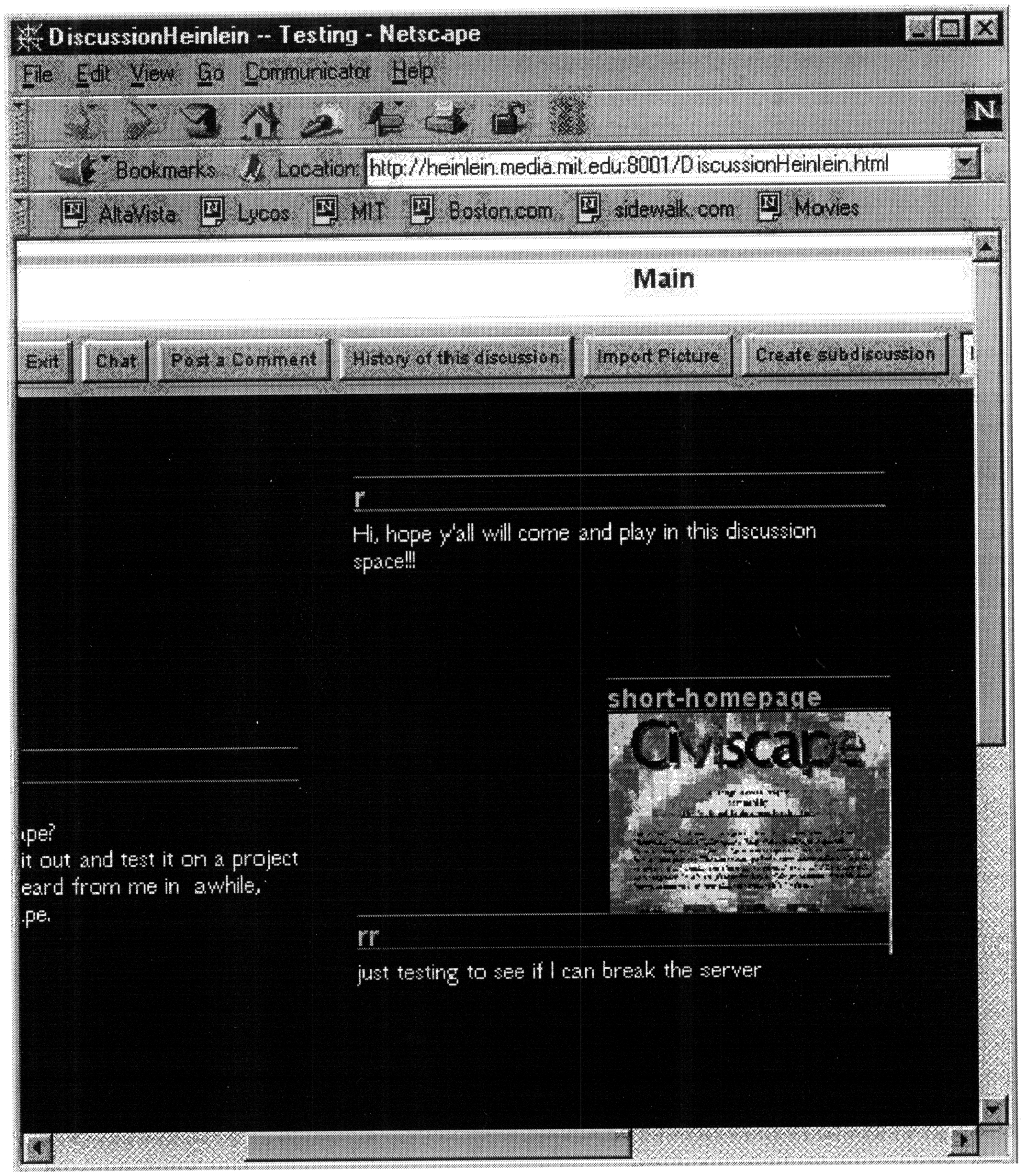
Over the past few years, the Internet has become increasingly popular and more and more widely used throughout the world. It provides access to enormous amounts of information and has tremendously enhanced the art of communication. It has become a source of entertainment and convenience for its users. Although the World Wide Web has been a great advancement in technology, it still has plenty of room for improvement. One weakness of the Web is the lack of interactive tools for artists and designers. Currently, very little software has been developed to facilitate discussions of three-dimensional models, or even two-dimensional images, by multiple users on the Web; the vast majority of the multiuser discussion tools currently on the Web are purely text-based. The Civiscape Project in the Media Laboratory at MIT aims toward developing tools for artists and designers that allow graphical discussions and online collaborative works to evolve.

## 1B. The Graphical Discussion Tool

One of the oldest projects of Civiscape is the Graphical Discussion Tool (GDT), a Java applet that allows discussions in a window-based environment over the Internet. After logging in to GDT, users can choose to enter any ongoing discussion or create a new discussion room. Once inside a discussion room, users can import image files or post comments to further the discussion. Each image and posted comment is embedded in a window that appears on each user's desktop (see Figure 1). These windows can be moved by clicking and dragging on the title bar. The contents of discussions are preserved between login sessions; so users do not necessarily need to log in simultaneously to carry on a discussion. Additionally, to better understand a discussion, users can pop up windows that display the history of the activity in a discussion room, as shown in Figure 2.

GDT also supports semi-realtime chatting. When a user opens a chat window, a receiving window pops up on every other user's desktop; the titles of these windows are the user name of the chatting user. The chatting user can send messages through a window that pops up on his own desktop. Any messages sent by are displayed in the receiving windows. Thus, for two users to have a one-to-one chat, both users need to open up chat windows.

The arrangement of the imported images and comments on the desktop is the same for all users. In other words, if one user moves a comment window or image window, that window moves on the desktops of all users. This decision was made so that posted comments would be easier to follow. If a user enters a discussion and several new comments have been posted, figuring out what each of them refers to may be difficult unless if they are placed next to the comment or image to which they is related. So when


DiscussionHeinlein -- Testing - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://heinlein.media.mit.edu:8001/DiscussionHeinlein.html
AltaVista Lycos MIT Boston.com sidewalk.com Movies
Main
Exit Chat Post a Comment History of this discussion Import Picture Create subdiscussion
short-homepage
Civiscape

r

Hi, hope y'all will come and play in this discussion space!!!

pe?
it out and test it on a project
eard from me in awhile,
pe.

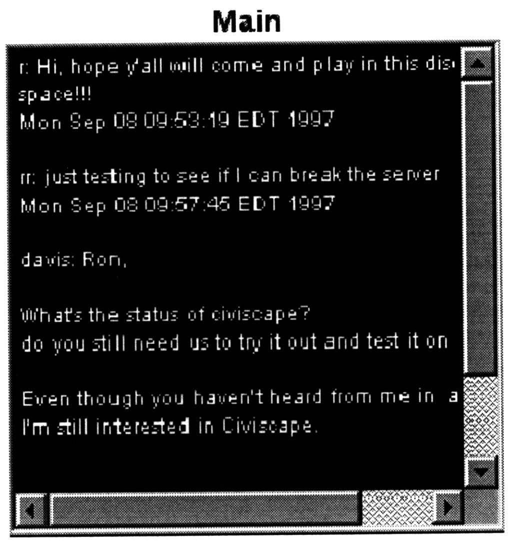
rr

just testing to see if I can break the server

*Figure 1: Snapshot of a small discussion in GDT*

a new comment is posted, the user who posted it is expected to move it to an appropriate place on the desktop. Although this setup requires some cooperation from users, not doing so can cause lots of confusion, as was discovered when GDT was alpha-tested. Note, however, that this shared layout does not apply to history windows and chat windows.

New discussion rooms can be created at any time, as well as rooms for discussing subtopics of an already existing room. From here on, these rooms that are used to discuss


Main

r: Hi, hope y'all will come and play in this dis
space!!!
Mon Sep 08 09:53:19 EDT 1997

rr: just testing to see if I can break the server
Mon Sep 08 09:57:45 EDT 1997

davis: Ron,

What's the status of civiscape?
do you still need us to try it out and test it on

Even though you haven't heard from me in a
I'm still interested in Civiscape.

*Figure 2: Snapshot of the history window of a discussion called "Main". Each comment is begun with the speaker's user name, followed by his comment. So the first speaker is r, followed by rr followed by davis. The history window's text can be viewed further using scrollbars.*

subtopics will be referred to as subdiscussion rooms, and the room of which they are a subdiscussion will be referred to as their superdiscussion room. Navigation from room to room is achieved through a menu at the top of the desktop. This menu contains a list of all subdiscussion rooms and the superdiscussion room of the current discussion room, as well as the home room, which is the superdiscussion room of the most general discussion rooms. (The home room is the only room that does not have a superdiscussion) When a user enters a different room, all the windows are cleared off his desktop, and the comments and images of the new discussion room appear.

Currently, GDT is being tried out by a small number of people throughout the nation who are interested in using it. Their feedback will be used to make improvements in GDT in the future.

## 1C. Design Discussion Space

Although the Graphical Discussion Tool is a step toward the goals of the Civiscape Project, it still falls well short of fully supporting online graphical discussions. GDT only allows discussions about two-dimensional images over the Internet. The application presented in this paper, known as the Design Discussion Space (DDS), expands on GDT by conducting part of its interface through a three-dimensional environment. On top of allowing importation of images, DDS supports importation of three-dimensional objects, and in the future, maybe even video clips and audio clips. Each discussion is conducted in its own separate three-dimensional space, but some of the interface still takes place in a window similar to GDT.

DDS is programmed in Java 1.0 and VRML 2.0, which stands for Virtual Reality Modeling Language. Originally, VRML (version 1.0) was a simple markup language for creating three-dimensional worlds. One could position spheres, cubes, and other shapes of any color and size to put together their worlds. The programmer did not need to worry about how to display the curvature, shininess, or transparency of the shapes in the world; all of those calculations were performed by the VRML browsers. In VRML 2.0, programmers can manipulate the virtual world and make it interactive using pieces of Java or JavaScript code. Because Java is more powerful than JavaScript, and because GDT was written in Java, DDS uses Java to enhance its virtual environment.

### 1D. Problems Encountered

Because Java and VRML 2.0 are such new programming languages, bugs in the languages and browsers are still being fixed. In the meantime, DDS needs to be implemented to work around these bugs. Although DDS could be programmed around most of these bugs, it could not be programmed around all of them. Thus, DDS is not complete, and can not be complete until some of the bugs in the programming language are fixed. (See Section 5A (*Bugs in the VRML Browsers*) for more details.)

The biggest hinderance to the development of DDS was the lack of VRML browsers that supported Java. When DDS first started, SGI's Cosmo Player was the only such browser, and it was still being beta tested. A non-beta version eventually came out, but this version did not support Java; therefore, development continued through the beta version. Unfortunately, once it expired, no further progress could be made until version 2.0 beta came out. Although version 2.0 beta was supposed to support Java, there were so many bugs in the browser that it also could not run DDS. No other (unexpired) browser has yet been found that can run DDS.

Currently, about ninety percent of the design presented here has been implemented and tested. Another seven to eight percent of untested code has also been written. The portion that remains to be implemented are described in Section 5B, *Future Implementations of DDS.*

## 2. RELATED WORK

Onlive! Technologies has developed a product called Traveller that allows people to talk by voice online in a three-dimensional virtual world. Users communicate through avatars, two- or three-dimensional icons used to represent each user. Sites viewable through Traveller include a Monday Night Football chatroom, a virtual casino called

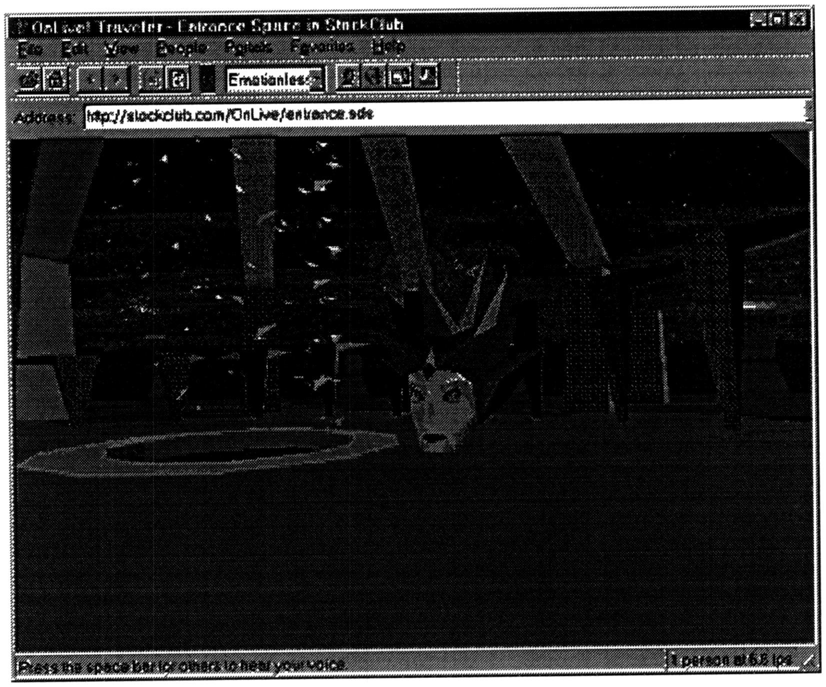

OnLive! Traveler - Entrance Space in StockClub
File Edit View People Robots Favorites Help
Emotionless
Address: http://stockclub.com/OnLive/entrance.sds
Press the space bar for others to hear your voice.
1 person at 6.8 fps

*Figure 3: Screen shot of Onlive! Traveller.*

Virtual Vegas, and Virtual Trading Floor, a world through which users can talk to investors from around the world.

Virtus Corporation has developed some software for creating virtual reality worlds. WalkThrough Pro and 3-D Website Builder both use primarily graphical interface to allow users to create virtual environment on the Internet.

TerraVirtua Design maintains a web site that lists lots of software related to virtual reality on the Internet. This software ranges from avatar chatting to virtual world development. These lists can be found through *http://www.terravirtua.com/vr/*.

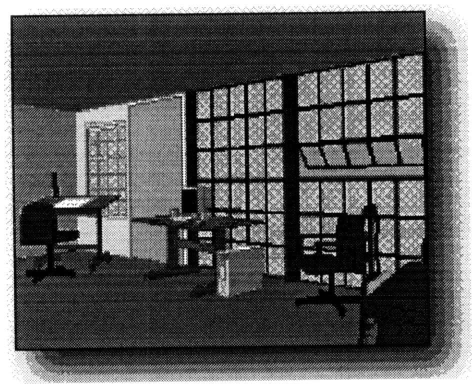
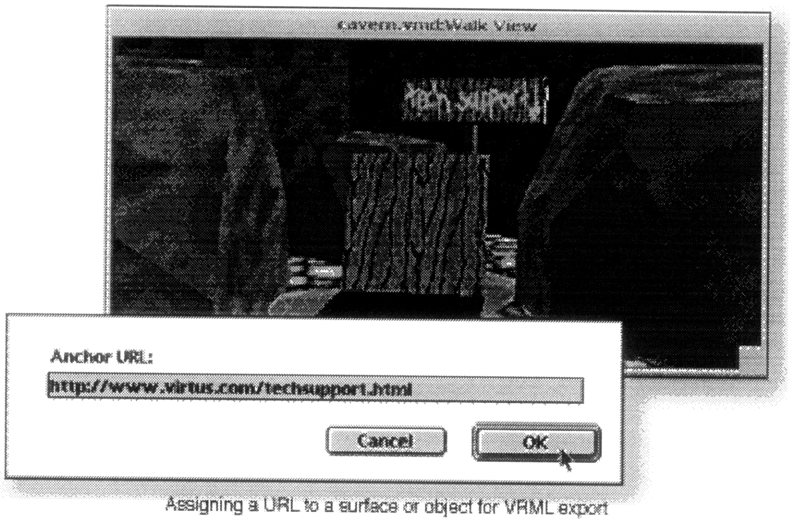


*Figure 4: Screen shot of Virtus WalkThrough Pro.*



Anchor URL: http://www.virtus.com/techsupport.html

*Figure 5: Screen shot of Virtus 3-D Website Builder.*

## 3. PURPOSE

The Design Discussion Space allows online collaborations on design projects and discussions about two- and three-dimensional objects in a three-dimensional environment. It is intended to be a more advanced version of the Graphical Discussion Tool, and will hopefully eliminate some of the problems inherent to two-dimensional envrironments, as well as bring in some features that GDT is missing.

First of all, three-dimensional objects and video clips can not be imported in GDT. Even if someone imports a set of two-dimensional snapshots of different viewpoints of a three-dimensional object, one can not always comprehend all of the details of the object from those snapshots alone. Additionally, generating multiple views of a three-dimensional object would be a hassle for users. As for movies, Java's current graphics package does not currently support MPEG movie files. So the only way to import movie clips would be to import a series of two-dimensional images, which again would be a hassle for users. Fortunately, VRML does support MPEG movie files, and, of course, three-dimensional objects.

Another missing feature of GDT is discussion previews; the only way to view the contents of a discussion is to actually enter it. Although the lack of previews may seem not to present any problems, having previews should save users some time. Occasionally, after a user enters a room, he may discover that the discussion does not quite cover a topic that interests him; so he exits the room to search for another one. However, each time the user changes rooms, his machine contacts the server and takes time to change the contents on his desktop. Thus, the entering and exiting of the discussion not only wastes the user's time, but also puts an unnecessary burden on the server. DDS will implement a preview feature to prevent such problems.

A third problem of GDT is that the desktops have a fixed size; so if too many comments are posted or too many pictures are imported, the desktop may become cluttered. Although GDT could be modified so that the desktop size can be manipulated, once a discussion gets large enough, its objects will not all be able to fit onto one screen; therefore, users will be forced to use scrollbars. Thus, locating a particular area of a discussion may become difficult for users. With a three-dimensional environment, one can choose the appropriate position in which to stand in the virtual world so that he can view everything he wants. Additionally, VRML worlds have a feature called viewpoints; when the user chooses a viewpoint from the viewpoint menu, he is brought to a particular position in the VRML world, as defined by the VRML file. In DDS, viewpoints could be defined so that users can go to imports, previews, etc., simply by choosing the appropriate item in the viewpoint menu.

# 4. APPROACH

## 4A. Overview

The process of creating DDS can be broken up into a number of large design problems. Some of the design issues were encountered when GDT was being created, while others were brought about by the extra third dimension provided by DDS. The major design issues that were solved during the creation of GDT are the following:

- *Interface.* The interface through which the user interacts with the environment should be easy to use.
- *Communication between server and client.* A solid, extensible protocol of communication between the server and client needs to be set up.
- *Logging history.* A system needs to be created to record everything that happens in each discussion room, from posting comments to creating discussion rooms. These logs need to be accessible to users at any time.
- *Logging in.* A secure system of recording user passwords needs to be designed, as well as an effective method for logging in.
- *Exiting.* The application needs to handle both normal and abnormal exiting from DDS. Abnormal exiting includes client machines crashing and browsers being closed unexpectedly.

Here are the design issues that were new in DDS:

- *Discussion room layout.* Designing the layout involves deciding where to place previews, imports, comments, and a chatting environment.

- *Navigation between rooms.* With a three-dimensional interface, a new method of navigation is necessary.
- *Subdiscussion previews.* To aid navigation, DDS implements previews for subdiscussions.

## 4B. Interface

### I. GDT

In GDT, all interface occurs in a windows-based environment embedded in a web browser. As shown in Figure 1, a panel of buttons is laid out across the top of the desktop, and the rest of the desktop is used for windows. When the user hits a button to import an image, post a comment, or create a new discussion, a window pops up asking for the appropriate information. (See Figure 6 for example.) After the user enters the information, the appropriate course of action is taken.

When a user opens a chat window, a sending window appears on his desktop, as displayed below, while a receiving window appears on the desktop of everyone currently logged into the same discussion. To have an interactive chat, one or more of the other users must also open a chat window through which they can type. Chatting can not be directed at particular users; if a chat message is sent, it can be read by any user logged in. Also, messages are only sent after the user hits the "Send" button (see Figure 7).

When a user opens a history window, a window appears that asks the user to choose a discussion or user name, or both. If just a discussion is chosen, a window appears, as shown in Figure 2, that contains all posted and chatting comments that have been made within the chosen discussion. If just a user name is chosen, that user's comments and chats are displayed. If both a discussion and a person are chosen, only the comments and chats made by the chosen person in the chosen discussion are displayed.

To navigate between discussion rooms, there is a menu that lists all subdiscussions and the superdiscussion of the current discussion, as well as the home discussion room. When the user chooses any of these rooms, the desktop is cleared, and all of the imports and comments that are in the chosen discussion room appear. Note that history windows and any window used to send chats are closed upon changing discussion rooms, as well as any prompting windows that appear when someone wants to import a picture, post a comment, etc. Therefore, after a user changes rooms, only windows for imports and posted comments are left on the desktop.

### II. DDS

When DDS expanded GDT into three dimensions, some changes needed to be made. One alternative was to entirely get rid of the windows interface, and have all of the interface through the three-dimensional environment. However, as the user navigates through the 3-D world, he should be able to have some sort of control panel available at all times. The easiest way to have such a panel is to create a separate window or frame

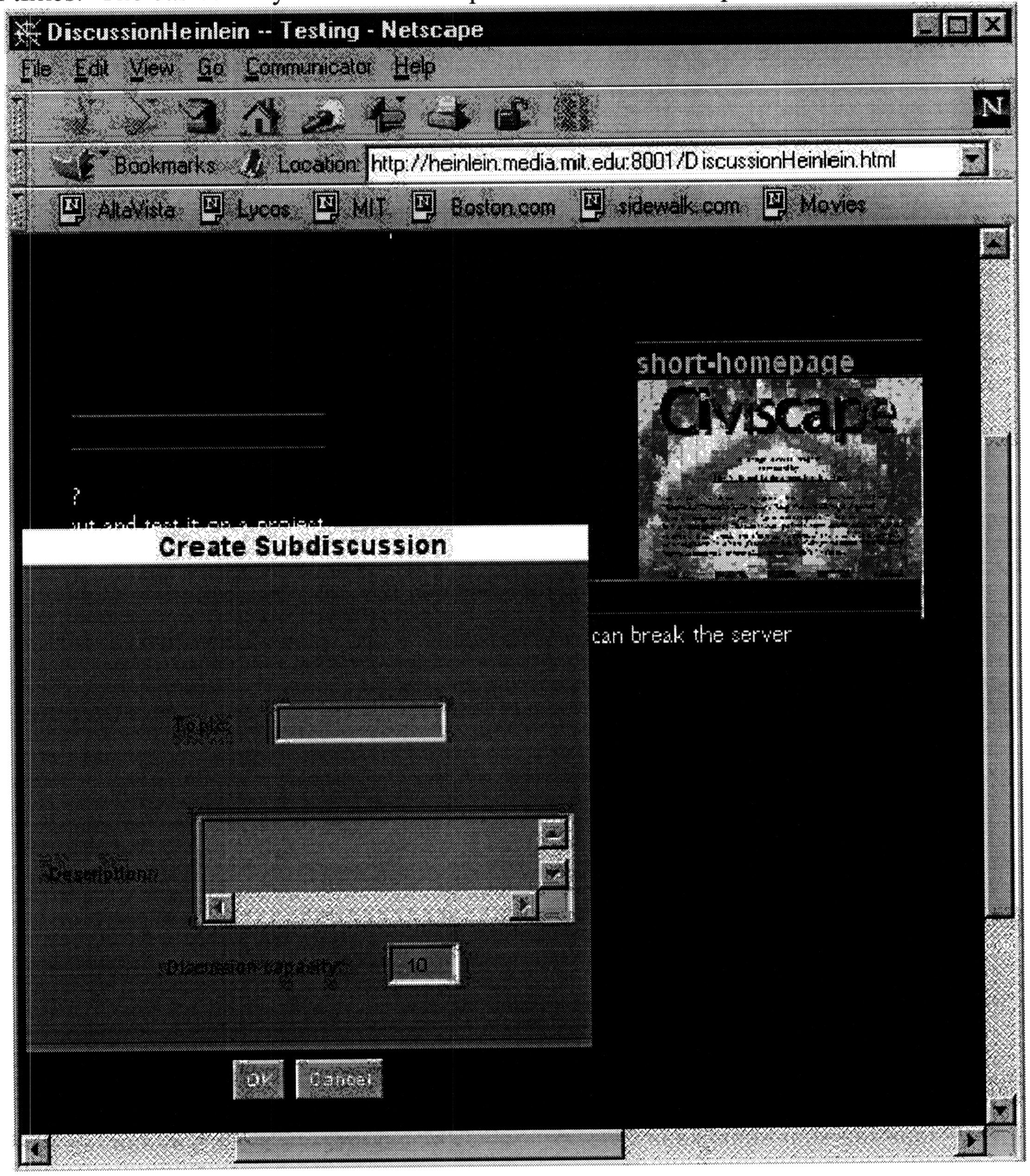

DiscussionHeinlein -- Testing - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://heinlein.media.mit.edu:8001/DiscussionHeinlein.html
AltaVista Lycos MIT Boston.com sidewalk.com Movies
short-homepage
Civiscape
can break the server
Create Subdiscussion
Topic:
Description:
Discussion capacity:
10
OK Cancel

*Figure 6: A prompting window, popped up when the user presses the "Create Subdiscussion" button on the button panel.*

that doesn't change as the user moves through the 3-D world. Thus, DDS has two windows of interface: One for the 3-D VRML world, and one to type in information, display histories, chat, etc. We will call the latter the *prompt frame* from here on. Figure 8 shows a snapshot of both windows of interface.

The prompt frame essentially uses the same interface as GDT, with a few small changes. One change is that the button panel in GDT has become a menu bar in DDS,

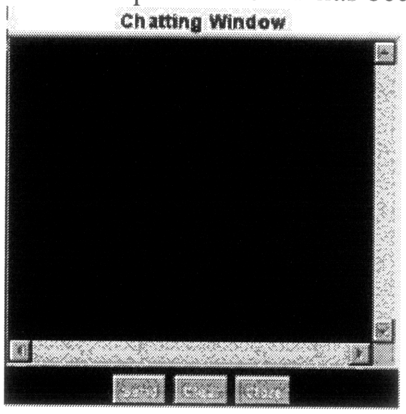


*Figure 7: Example of an empty window used to send chat messages.*

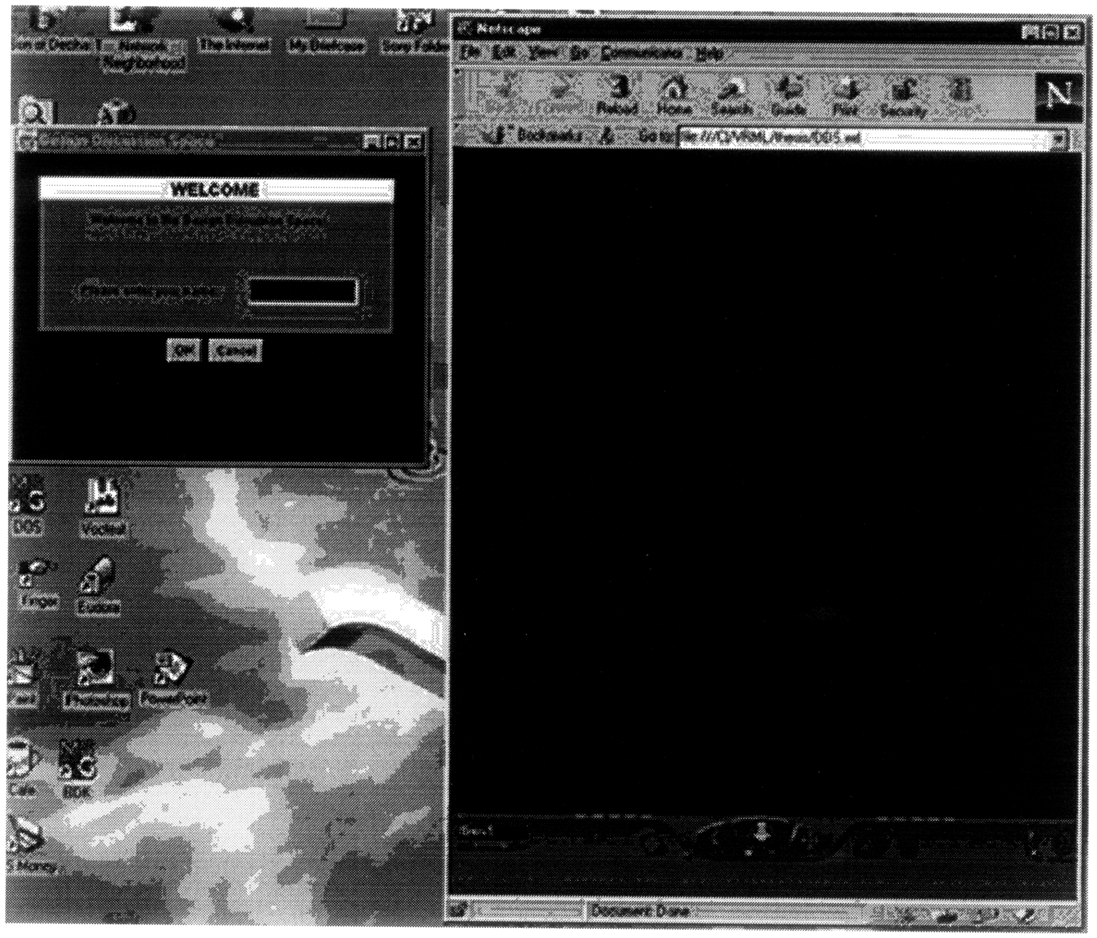


*Figure 8: A snapshot of the login screen for DDS. The left window is the prompt frame, and the right is an empty VRML world, embedded in a Netscape browser. After the user*

*logs in, objects are added to the VRML world, and a menu bar appears in the smaller window.*

which makes the panel more aesthetically pleasing and more organized. Another change involves posted comments and imports, which are placed in the three-dimensional VRML world instead of on the two-dimensional desktop. Chatting has also been changed slightly: Rather than only being able to chat with people who are currently in the same room, in DDS, one can chat with anyone logged in anywhere on DDS. History windows still are displayed as before in the prompt frame. The prompt frame still contains a menu for navigation between rooms, but users may also use the VRML world to navigate. This feature will be explained in Section 4D, *Navigation Through the VRML World.*

## 4C. VRML World Layout

The layout of the VRML world should be expandable for new subdiscussions and imports, but at the same time, easy to navigate around. Additionally, the imports should be laid out so that users can easily understand the discussion. For example, if one comment is a response to another, those comments should be placed together so that users viewing the discussion can see that they are related.

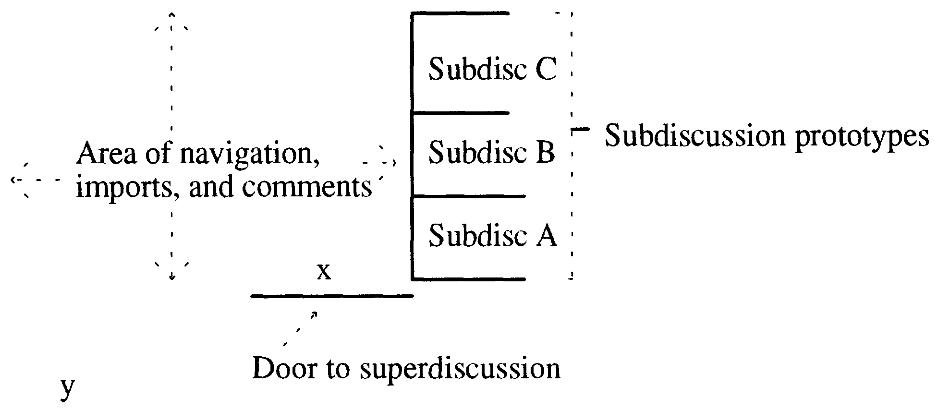


*Figure 9: General Layout of Discussion Spaces in DDS.*

Figure 9 is a diagram of the general layout of the room. Each subdiscussion is represented by a three-dimensional room without a back wall. The user can view these subdiscussion prototypes from the navigation area because the walls of the subdiscussion prototypes are transparent. Subdiscussion prototype will be discussed further in the next section, *Navigation through the VRML World.* The navigtion area includes all imports and comments. When a user first enters a room, he starts out in front of the superdiscussion door (where the "x" is in the diagram), facing the navigation area. Figure 10 shows a screen shot from point x, and Figure 11 show's a bird's eye view from point y.

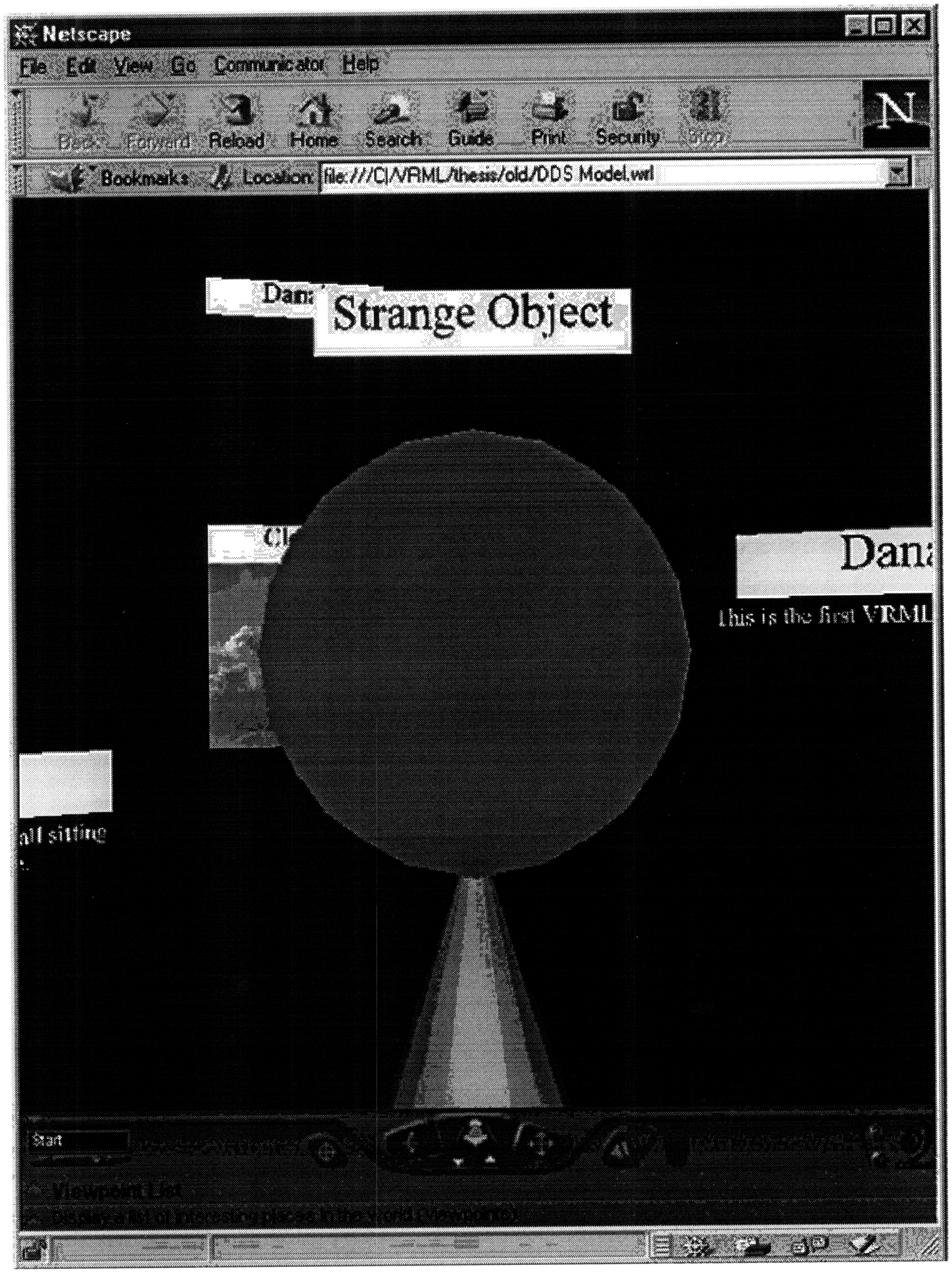

Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: file:///C|/VRML/thesis/old/DDS Model.wrl
Strange Object
Dana
This is the first VRML
Viewpoint List

*Figure 10: Example of a discussion room in DDS. This is the viewpoint the user sees when he first enters the discussion.*

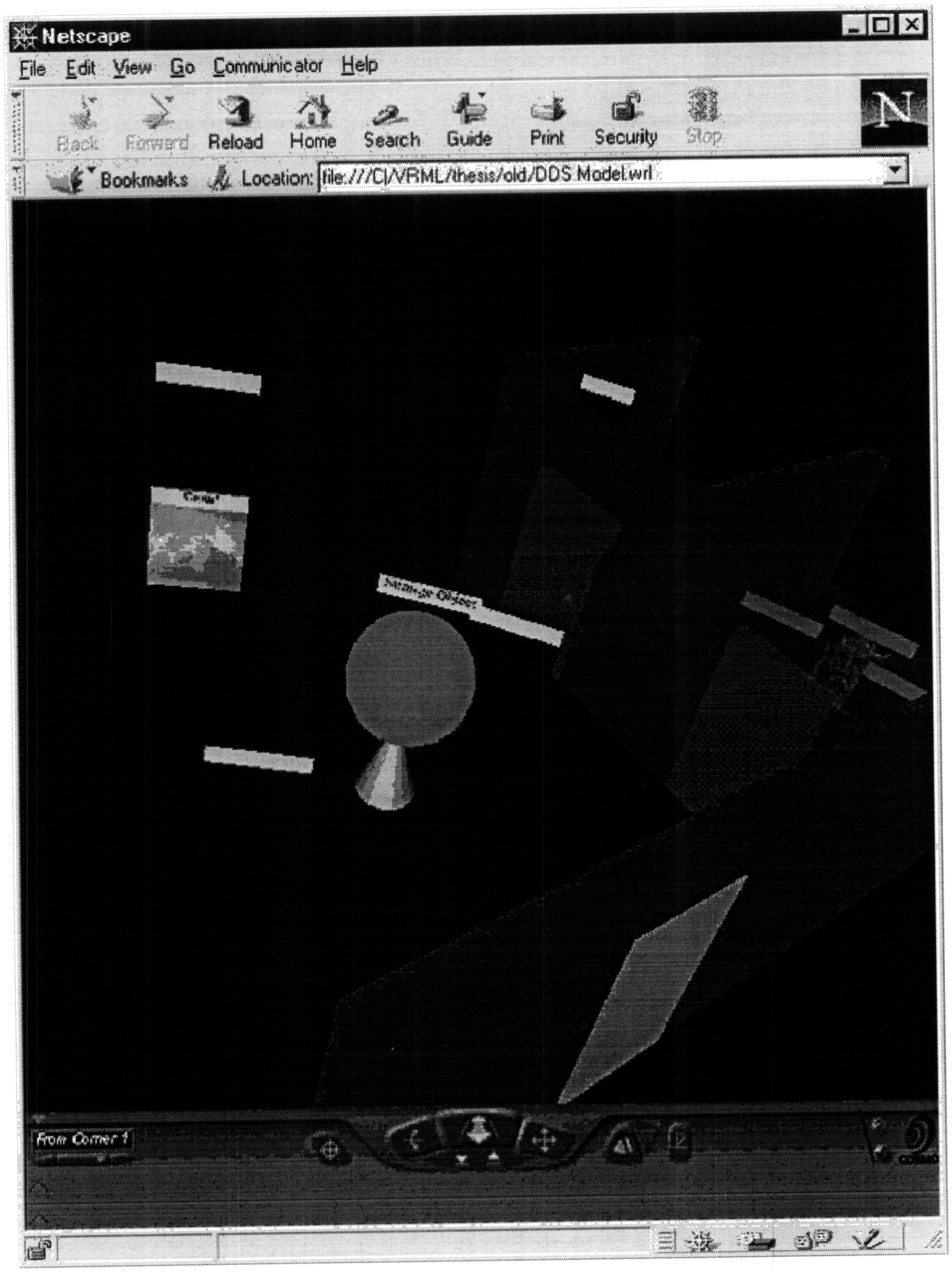

Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: file:///C|/VRML/thesis/old/DDS Model.wrl
From Corner 1

*Figure 11: A bird's eye view of the same discussion.*

A person can walk around the navigation area using any VRML 2.0 Browser, and examine different subdiscussions, imports, etc. Navigation through the VRML world is aided by a feature in VRML called viewpoints. When a user chooses a viewpoint, he is moved to some specified location in the world. For example, in DDS, if the user chooses the "To Superdiscussion" viewpoint, no matter where he has wandered off in the VRML world, he will be moved back to point x in the diagram, facing the superdiscussion door. Thus, should a user want to go to a door or object immediately, he can simply choose one of the viewpoints. Figure 12 shows a screen shot with the viewpoint menu displayed. In DDS, there is a viewpoint for each import and subdiscussion, as well as one for the initial position, and for the superdiscussion door.

When a subdiscussion is added to the VRML world, its prototype is placed next to the last subdiscussion prototype. The first subdiscussion prototype is placed next to the superdiscussion door. As an illustration, referring back to Figure 9, the first subdiscussion prototype would be placed at the position marked "Subdisc A." When the next subdiscussion is added, its prototype will be placed at the position marked "Subdisc B," and the next at "Subdisc C."

When objects are first imported into a discussion, and when comments are first posted, they is placed near the position marked "x." The user can then move the object by clicking on the title bar of the import. When the title bar is clicked, a set of small three-dimensional axes appear. When the user clicks on one of these axes, movement for the object is locked in the direction of the axis. The user can then move the object by clicking on it and dragging. If a user want to move the object diagonally, he would need first to click on one axis, move the object in that direction, then click on another axis, and move the object more in the second direction. Once the user is done moving the object, he can click on the title bar again to make the axes disappear. Although this procedure may be a little roundabout, it is easy for users to understand and use, especially considering that the interface for the three-dimensional world is limited to a two-dimensional computer screen. Figure 13 shows the moving axes of an import, with one of the axes highlighted.

The layout each room is recorded on the server side. These files contain the positions of all of the imports and comments in the room, as well as the size of the room. The size of the room is measured by finding the most extreme points of the imports and comments in the room. In other words, it finds the dimensions of the smallest rectangular prism that can enclose all objects in the room. For a detailed description of layout files, see Appendix G, *Layout File Format.*

Every time a user adds an import or posts a comment, a message is sent to the server. The server then adds the new object to the layout file and sends messages to the machines of all users currently in the same room. On the client side, the new object is added appropriately to the VRML world. When a user moves an object, a message is also sent to the server, which in turn updates the layout file and sends a message to the appropriate client machines. As in GDT, all users share the same layouts.

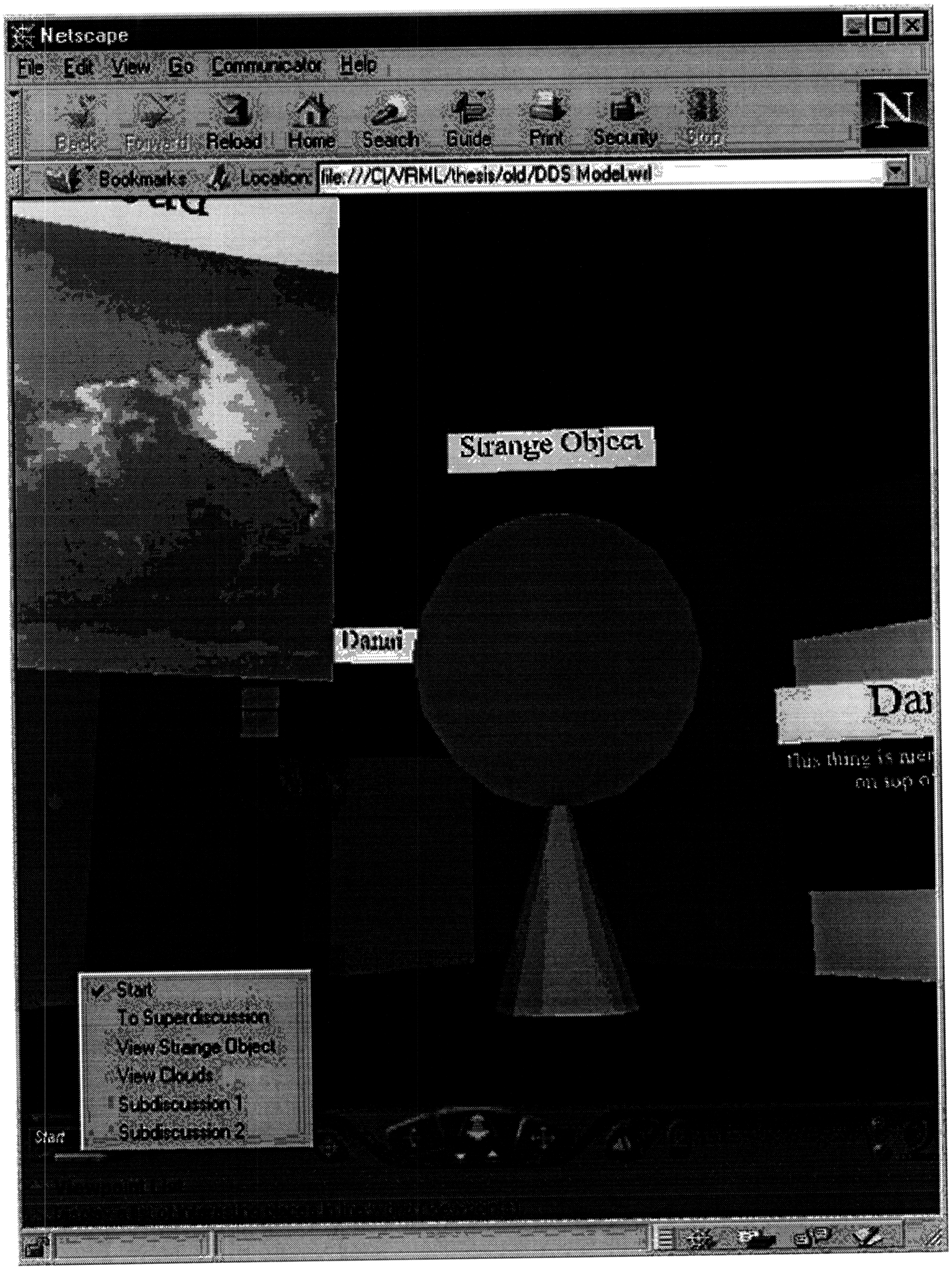

Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: file:///C|/VRML/thesis/old/DDS Model.wrl
Strange Object
Danni
Dai
Start
To Superdiscussion
View Strange Object
View Clouds
Subdiscussion 1
Subdiscussion 2

*Figure 12: DDS, with the Viewpoint menu popped up. Each import and subdiscussion, as well as the superdiscussion and the starting veiwpoint, appear on the menu.*

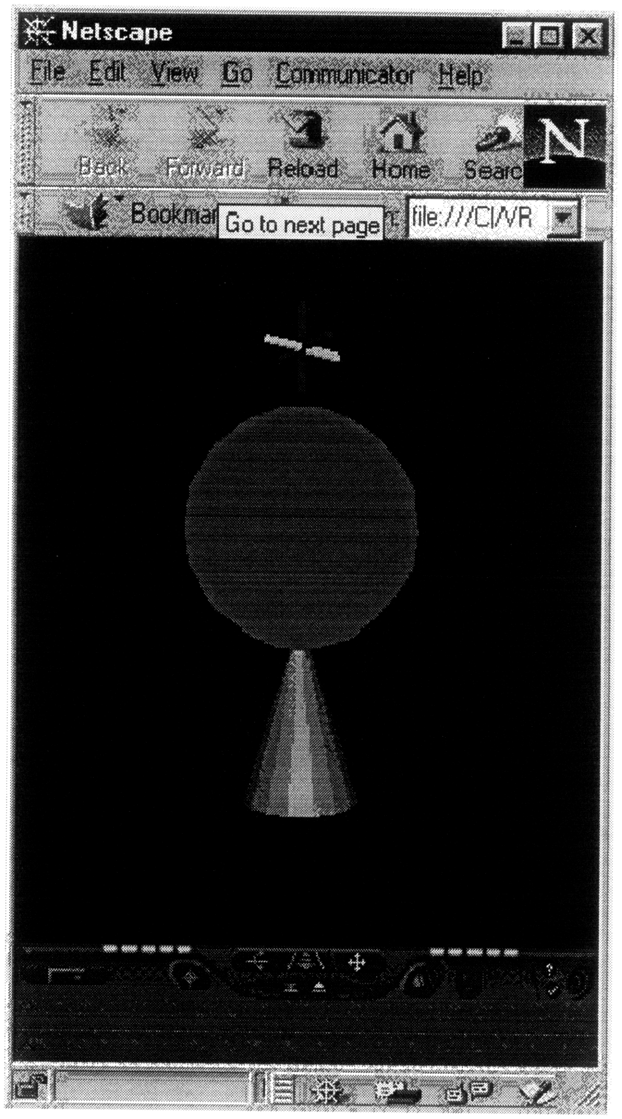

Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Searc
Bookmar Go to next page
file:///C|/VR

*Figure 13: A 3-D import, with axes for moving displayed. Currently, one of the horizontal axes is highlighted*

## 4D. Navigation through the VRML World

Although navigation can be performed through the prompting frame, as explained in Section 4B, DDS also allows navigation between rooms through interface in the VRML world. With navigation in the VRML world, one can preview a room before entering. As explained in the previous section, each subdiscussion is represented by a three-walled room in the virtual space. This room has transparent walls through which the user can view a prototype of the subdiscussion room. On the door of the room is the title and description of the subdiscussion. If a user decides to enter the subdiscussion, he merely needs to click on the door.

When the user clicks on the door, all of the objects are removed from the VRML world, and the server is notified that the user has changed rooms. In turn, the prompting frame is notified of the room change. Next, all of the objects in the new room (the subdiscussion) are placed into the VRML world, and the user is placed in front of the door to the room from which he just came, at point x in Figure 9. To return to the original room, the user can simply click on the superdiscussion door.

When the user clicks on a superdiscussion door, the same thing happens. Logically, the user should be placed in front of the door of the subdiscussion from which he just came (facing the navigation area), but actually, the user is placed at point x. Although this setup is physically illogical, it keeps the program more simple because the program does not have to figure out where to place the user; it simply always places it in front of the superdiscussion door.

## 4E. Communication

Currently, the communication is quite simple and straightforward. All messages between server and client begin with a string code, which tells the receiver what the purpose of the message is. Special bits are used to mark the beginning and end of messages. Untypable special characters are used to separate different parts of a message. (e.g., the string code from the rest of the message, or different arguments within the message.) The message system is very extensible: If more functonality needs to be added to the system, a new string code simply needs to be created; if the arguments for a given string code needs to be changed, it can be added without any trouble. For a complete list of string codes and parser characters, see Appendix E (*Message Protocol*) and StringCode.java in the code (Appendix H.)

Because of the multithreaded environment, especially on the server, the program must guarantee that messages are not sent simultaneously through the same channel of communication. Thus, a class was created to manage each socket to guarantee avoiding any such interference. The implementation of this class can be found in StreamManager.java in Appendix H.

The client side communicates through two sockets: one for the prompt frame, and one for the VRML world. Because the prompt frame and the Java code for the VRML world are two different programs, each one needs a separate channel of communication. The server pairs together the two sockets used by a given machine at login time. In the future, each client may communicate through one socket. See Section 5B, *Future Implementations of DDS.*

## 4F. Logging Discussion History

Logging the history of a discussion serves several purposes. Because the comments and images are laid out in the VRML world with no indication of the order in which they were created, users may find the history feature helpful because it displays an ordered list of comments. This list not only lets the user examine how the discussion progressed over time, but also helps him see how certain comments are related to each other. Or, likewise, one may investigate how another user's views have progressed over time by viewing that user's history. Logging is not only convenient, but necessary: If the server crashes, it needs some way to recover the content of the discussion. The easiest way to guarantee this recovery is to save everything that happens in the discussions to disk on the server side.

Every discussion room and user has a history file. The name of the history file is the discussion's title or the user's user name, with ".history" appended to the end. To guarantee that no two history file names will be the same, no discussion or user are allowed to have the same name.

History files are text files so that they are human-readable. The files are a series of records, each logging one particular action within a discussion. These actions include importing objects, posting comments, sending chat messages, and creating discussion rooms. Each record takes up a few lines, and is separated from other records by a line of special characters (see StringCode.java in the code, Appendix H). Every record starts with a string code to indicate type of action being logged (import object, post comment, etc.). The following two lines are the user and discussion involved in the logged action. The rest of the record is other information about the action. For details about the fields of each record type, see Appendix F (*History File Format*) and Recorder.java in Appendix H.

## 4G. Logging in

All passwords are kept in one file. The odd-numbered lines of the file contain the user names; the even-numbered lines contain their respective passwords. In other words, line 2 is the password for the user name in line 1; line 4 for line 3; etc.

When a user logs in, he is asked to enter his user name. People who have used DDS before should enter their user name, and people who have never used DDS can choose a user name at this point. After the user enters a user name, the client machine sends a message to the server. The server then checks whether the user name given matches a previous user's. If it does not, the server assumes the person is a new user and tells the client machine to ask him to choose a password. After the user chooses a password, the client sends the password back to the server, at which point the server updates the passwords file.

If the user name does match a previously used one, the server checks whether anyone currently logged in is using that user name. This check is performed to prevent two clients from using the same user name at the same time. If two clients are using the same user name, the server would get confused between them. More specifically, when a message is supposed to be sent to one of the clients, the server would not know which client to send it to. If a user logging in enters the same user name as someone currently logged in, the server tells the client machine to ask the user to enter a different user name.

If the user name entered matches a previously used user name, but no one is currently logged in under that name, the server tells the client machine to ask the user for the password, and sends it the correct password. Once the user enters the correct password, the client notifies the server to tell it that the user has successfully logged in.

Of course, some new users may happen to choose previously chosen user names, and some old users may mistype their user names. So, when the user is asked to choose or enter a password, he has the option of going back and retyping his user name.

## 4H. Exiting DDS

A user exits DDS by choosing the appropriate menu item in the prompt frame. When this item is chosen, a message is sent to the server, and the client machine closes the prompt frame. The server subsequently closes any chat window that might be open for the user, and records the room from which the user exited. When the user logs in again, he will automatically placed into the room from which he exited. Also, the server closes any network sockets connected to the client machine.

DDS must also, however, be able to take care of instances where the client's browser or computer crashes. If nothing is done in such a situation, the server will still think the client is logged in; so he will not be able to log in again. When a client machine is unexpectedly disconnected, the server is sent a signal. Upon receiving such a signal, the server kills any threads related to the machine, and behaves just as if the user has logged out. In other words, the user's chat window is closed, and the room in which he was last participating is recorded to disk.

## 4I. Subdiscussion Previews

If the subdiscussion prototype is too detailed, it may unnecessarily slow down the browser because of the extra computations needed for the details. On the other hand, if it is not detailed enough, it may not serve as an effective preview for the subdiscussion. Thus, we need to decide an appropriate level of detail for the prototype, and a configuration that allows the most information to be gathered without considerably slowing down the browser.

Because the primary goal of the previews are to show the content of the subdiscussion, the imports should definitely be included in the prototype. However, if they are shown at full size, then the prototype may be considerably large. If the prototype is too large, it does not serve as a useful preview, because understanding the basic content of the subdiscussion would require navigating around the prototype. Having a smaller prototype (but not too small!) would be more desirable, so that a survey of the subdiscussion can be made without much effort. Thus, the whole subdiscussion space is scaled down if necessary to fit into a three-walled room.

If the prototype gives users a view from ground level, some objects may be difficult to see, partly because they are so far away, and partly because they are hidden behind other objects. Thus, the prototype is turned so that the user, when looking through the transparent wall, sees the subdiscussion as if he were looking directly from above. Figure 14 shows the prototype for the discussion room displayed in Figure 11; this prototype would be displayed if the user were in the superdiscussion of the room shown in Figure 11. The objects are arranged as if the room were being viewed from straight above, but, as you may observe, they have been turned up so that they are being viewed directly from the front. Thus, one can see the general layout of of the room, but at the same time, get a good view of the imports in the subdiscussion.

In addition to showing the basic content of the room, the previews display comments to show how active the room is. The text of the comments are omitted for two reasons. First of all, because the prototype is scaled down from the subdiscussion's actual size, the text would be difficult to read anyway. Additionally, text in VRML worlds considerably slows down browsers because of the number of calculations they require to be drawn; since the prototype serves merely as a preview, the text is not necessary. For the same reason, title bars of imports are omitted in the prototype.

To help users understand the them of the subdiscussion, the door to each prototype displays the title and description of the room.

# 5. FUTURE WORK

## 5A. Bugs in the VRML Browsers

Because VRML 2.0 is such a new programming language, there are very few browsers that support it. Because all of these browsers are still in their developmental stages, they still contain many bugs and limitations. Therefore, implementing a complete version of DDS is currently impossible, and the project can not proceed further until a debugged, complete version of a VRML 2.0 browser is finished.

During testing stages, SGI's Cosmo Player was the only browser that supported VRML 2.0. However, it was only written for Windows and SGI platforms. Additionally, neither version was complete: In Windows, several types of nodes were not yet supported, including Text nodes; thus, no words can be seen in the windows version of DDS. On the SGI platform, Java was not supported, which means that the world can not

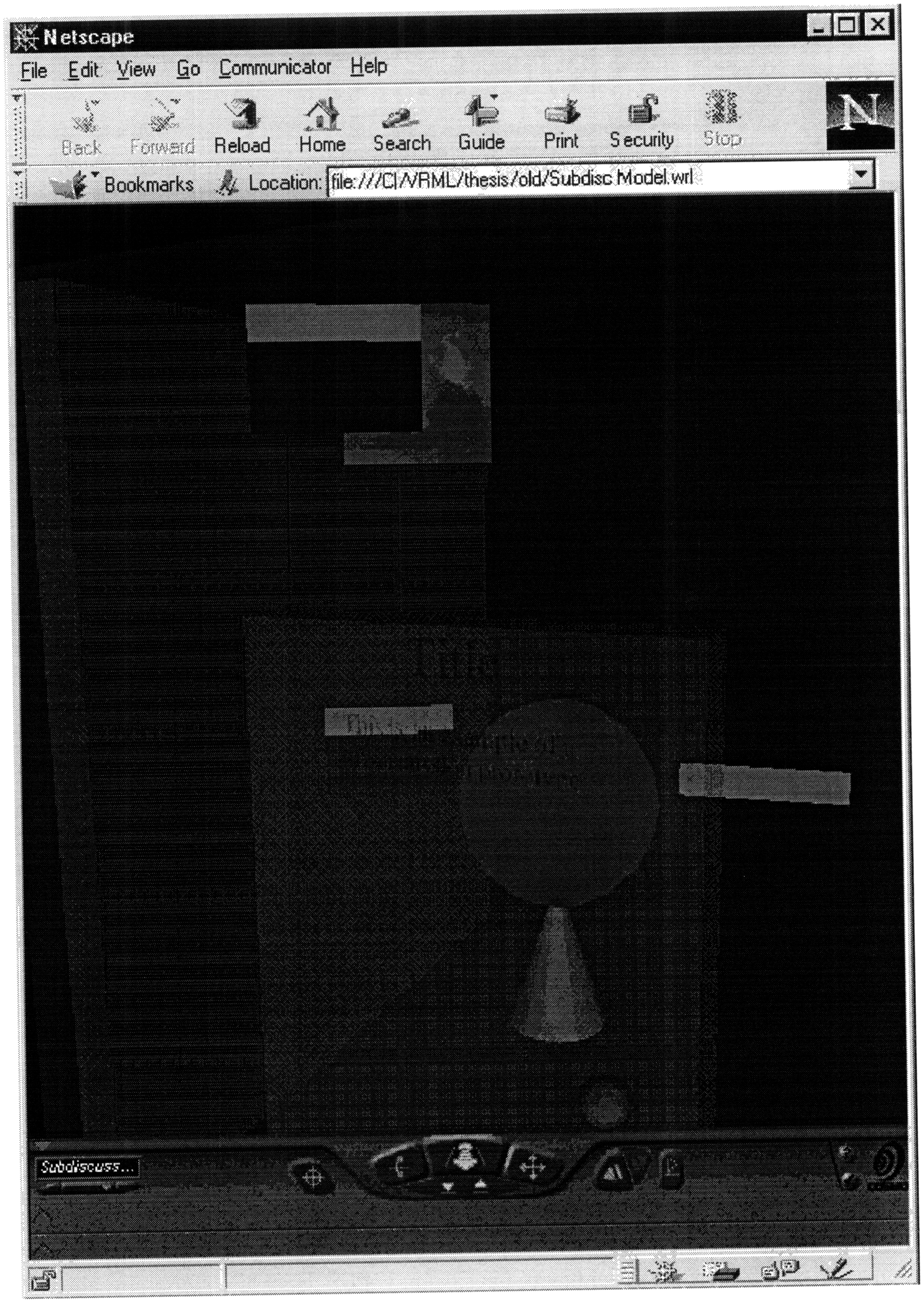

Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: file:///C|/VRML/thesis/old/Subdisc Model.wrl
Subdiscuss...

*Figure 14: Example of a subdiscussion prototype.*

be very interactive, and no network communication is possible at all. Thus, all testing was done in Windows, and occasionally, some tests were performed on SGIs occasionally to size text properly.

However, these browsers were beta versions, and therefore expired after a while. When the first non-beta version 1.0 came out, the Windows version had come to support Text nodes. However, neither the Windows nor SGI versions supported Java at all; so testing was continued on the beta version until it expired. Once it expired, no further testing could be done until version 2.0 came out, which would supposedly support Java. However, version 2.0 contained many bugs which effectively left Java still unsupported. As of now, version 2.0 is still the most recent usable version of Cosmo Player.

Although other VRML 2.0 browsers exist, none have been found that fully support Java; thus, no testing has been performed on DDS since the beta versions of Cosmo Player 1.0 expired. In the meantime, compilable (but untested) code for removing discussion rooms and improving graphical interface have been written. When a VRML 2.0 browser comes out with Java functionality, this code will be tested and debugged. Additionally, the code for placing text on comments, title bars, and doors of subdiscussion prorotypes needs to be retested, because DDS has had many changes since this code were last examined.

DDS also sometimes causes bus errors and crashes the browser. The reason for the crash has not yet been found; the times of the crashes are quite random and unpredictable. Although there may be specific pieces of code that cause the crash, Java is a garbage collecting language; therefore, memory management is not the responsibility of the user. On the other hand, the problem may not lie in the program or the compiler, but the browser, or maybe even the operating system. In any case, solving this problem is out of the control of the user.

Finding the size of imported images presents another problem. When an image is imported, the Java program is supposed to wait for it to get downloaded to determine the image's size. Although determining the size works fine in GDT, for some undetermined reason, DDS does not wait for the entire file to be downloaded before calculating the image's size; thus, the file size yielded is entirely incorrect. So, right now, all imported images are pasted on a canvas of the same size, regardless of actual image size, and stretched or squashed appropriately to fit that canvas. Although the sizing problem may be a bug in DDS, because the exact same code works in GTT, the true source of the problem has not yet been located.

One more problem concerning imports deals with URL address input. Currently, DDS can only detect whether a given URL address has a valid machine name. If the machine name is invalid, then the user is told so and asked to reenter the address. However, if the machine name is valid, but the file name is not, DDS tries to import the file, and, of course, an error occurs. This bug stems from Java: An exception is supposed to be thrown when the address is invalid, but Java only detects invalid addresses when the machine name is invalid. So if the machine name is valid, but the file name is not, no exception is thrown. As of now, no method has been found to identify invalid file names.

Another major missing feature in DDS is the movement of objects (see Section 4C, *VRML World Layout*). Although code has been successfully written and tested for moving individual objects, the VRML browsers currently available will not allow moving axes to be added dynamically: Since the movement functionality requires a separate script for each object, every time a new import or comment is added to the discussion space, a new script also needs to be added; unfortunately, the browsers currently available do not support the dynamic addition of scripts. To get around this problem, DDS currently arranges all objects in a line in the navigation area. Although the objects can not be moved, this solution is the most practical way around the problem until the browsers are fixed.

Because work is being done on VRML and browsers as we speak, this list of problems may be (and hopefully will be) outdated in the very near future. In fact, some of them may have already become outdated in the time it has taken to write this report. But surely, as the project develops further, more problems will arise. In the meantime, solutions need to be found to work around these temporary problems, so that the project can advance as quickly as possible

## 5B. Future Implementations of DDS

### I. Overview

DDS is far from complete; it still falls well short of software usable by the public. Many holes still remain in security, as well as bugs related to concurrency. Several changes could also be made to improve DDS's performance in speed and memory management. And although DDS supports a great deal of functionality now, its capabilities should be expanded in the near future.

### II Security

The most important bug to fix deals with security. Currently, anyone can read passwords being sent over the Internet; DDS utilizes no encryption methods to protect the passwords. Although the lack of encryption makes debugging easier, the final product should implement some form of security.

DDS also heavily trusts users not to carry out any malicious actions that could crash the server or disturb the discussion. Right now, the only way to identify users is through their user name; thus, the true identity of the person is never known. So, any user could post a million meaningless comments -- which would slow down the server, waste disk space, waste memory, etc. -- and no one would ever know who did it because only the person's user name is attached to those comments. Or, a user could simply move around the objects in a discussion so that the discussion is hard to comprehend. For example, he may move the main object of discussion to a distant point, while spreading the comments out all over the discussion room space.

One solution would be to get the true identity of the user. Perhaps DDS could record the e-mail address from which the user is accessing the DDS web page. Of course, one does not need e-mail to access DDS; for example, one may reach DDS through Netscape Navigator on a Windows 95 platform. So maybe DDS could only allow access to users whose real identity can be determined. Of course, this solution would not actively prevent users from performing malicious actions, but if one does perform one, he could easily be identified.

Another security measure could be to limit the number of comments a user can post in a particular day. The same rule can be applied for imports. At the same time, a human managing DDS (on the server side) could monitor the daily comments and imports to make sure no one is doing anything malicious. Additionally, DDS could be augmented so that it records the person who moves an object in a discussion.

## III. Concurrency

Concurrency is another major issue. Although concurrency bugs only elicit harmful effects very occasionally, the program should be implemented so that there is no chance of failing. However, concurrency bugs are hard to detect, and even for concurrency problems that have been considered, testing to confirm that the code prevents those problems is very difficult.

One unresolved problem occurs when a machine crashes. If the server is writing to disk when it crashes, then a partially written file will result. So, when the server is restarted, it may not behave properly when it tries to read that file. One possible way around this problem is to have two copies of every file. When the server wants to write to a file, it only writes to one of the copies, and when it is done writing, it marks that copy as being the most current one. After that, it updates the other file. Thus, if the server crashes in the middle of a file write, there will be at least one uncorrupt version of the file, either in the state before the writing, or in the state after the writing.

With multiple threads running on the server side, the server may sometimes try to write to the same file from two different threads at the same time. In the future, DDS should implement a system for locking files so that writes to the same file are always done serially; every time a thread want to write to a file, it must lock it, and no other thread may write to the file until that lock has been released.

Although only two examples of concurrency problems have been presented, there are surely many more. Unfortunately, it may be impossible to conceive of all of them until the system is released for public use, and even then, solving concurrency problems will be difficult.

## IV. Memory Management

Although computers are being built with more and more memory nowadays, different users use different types of machines; so it can not be guaranteed that every user will have enough memory to run DDS. Due to a limitation of resources, and the fact that VRML 2.0 only works on SGI and Windows platforms, DDS has only been tested on two different types of machines. Eventually, DDS will be able to be accessed from any type of machine. Naturally, when that time comes, DDS should be tested on other platforms, but in the meantime, as many bugs as possible should be eliminated.

Several solutions exist to insure that a user will not run into memory problems. First of all, DDS could utilize the user's disk space when his machine's memory is full. Of course, doing so may run into problems with network security. Another solution would be to always contact the server when the client's machine is running out of memory. Because the server stores all of the information anyway, this solution is clearly a possibility; however, network communication is much slower than accessing a local disk; so the first solution is definitely more desirable when possible.

Caching is also a potential performance improvement for DDS. If a user is navigating through DDS, switching between discussion rooms often, caching the discussion room layout would prevent having to download the entire layout every time a user enters a room, especially if the user had just entered the room earlier in the same session. The cache could be stored either in memory or on disk.

## V. Communication

One major fault in the design of the communication system is communication between the prompt frame and VRML world of the same user. For the prompt frame to send a message to the VRML world, and vice versa, it must send a message to the server, which subsequently processes the message and sends something to the VRML world. In other words, the prompt frame can never directly communicate with the VRML world.

The reason for this horrible design relates to a problem encountered in the early stages of DDS development: Previously, when Java was used to create new windows, the computer would crash. Naturally, that sort of behavior needed to be avoided; so DDS was designed so that it operated out of two frames in a web browser. Unfortunately, when a browser is divided into frames, each frame must run a separate Java program. Because the Java program for the prompt frame and for the VRML world were different, there was no way for one frame to communicate with the other without setting up a server on the client machine and communicating through the Internet. So, instead, all communication took place through the server.

Eventually, the Java bug was fixed; so windows could be created safely on any platform. So, the implementation of DDS was changed so that prompt frame popped up as an entirely new window, while the VRML world was still embedded within the browser. With this setup, both the size of the prompt frame and the VRML world could be adjusted easily and independently. Additionally, the prompt frame was no longer taking space away from the VRML world within the browser.

The next step would have been to fix the roundabout communication. Right now, the programs in the promptframe and VRML world are still separate; eventually, the VRML world should create the prompt frame. Thus, they would be able to communicate with each other without using the Internet, since they would actually be part of the same Java program. Unfortunately, before this change could be made, the version of Cosmo Player that supported Java expired.

## VI. Navigation between rooms

One feature of DDS which could not be implemented due to time constraints is a main menu that is displayed each time a user logs in. Currently, the user is automatically placed in the discussion room from which he exited DDS, which may not always be desired. The first time a user logs in, he is placed in the home discussion, called "Main." Currently, this discussion room can be used just like any other discussion room, although in actuality, it should only be used to create subdiscussion rooms, and not to import objects or post comments. Thus, a main menu would be a more logical interface. From this main menu, a user could enter the room from which he last exited, enter any of the major discussion rooms, or create another major discussion room. Users should also be able to view histories and previews of the main discussion rooms, as well as pop up Help windows.

The subdiscussion prototype is also far from complete. The current setup is not necessarily ideal for every discussion space; it makes assumptions about the layout of the subdiscussion. More specifically, it assumes that all imports and comments are laid out at the same horizontal level, and that they are well spread out. A more useful approach would be to have the prototype be a smaller replica of the subdiscussion that can be manipulated as the previewer wishes. For instance, he could rotate it, or zoom in and out on it. Of course, a prototype with so much functionality may slow down the browser; so there is a tradeoff between usefulness and speed. Another useful feature would be to write the last time anyone entered a subdiscussion on its prototype's door. This way, users can see whether the subdiscussion has been dead for a while.

The navigation between rooms can be made more logical if users, after passing through a superdiscussion door, are placed in front of the subdiscussion they just exited. Currently, when the user passes through the superdiscussion door, he is placed in front of the superdiscussion door of the superdiscussion, which may confuse some users.

Another possible addition to DDS is limiting access to certain rooms. Sometimes, users may desire to hold private discussions. Thus, the creator of a discussion room should be able to specify which people or group of people are allowed access to the room.

Naturally, the manager of DDS on the server side will also be able to view the contents of the discussion, but no clients who are not specified will be allowed to enter the room.

## VII. Imports

Right now, only images and three-dimensional VRML objects can be imported. In the future, hopefully, audio clips and video clips will also be allowed. The icon representing such clips would look like a VCR panel, with play, stop, pause, and restart buttons; these icons could be moved around the space just like any other imports. The main limitation of implementing these import types now is that VRML movies files and audio files do not work on all platforms.

Another limitation on imports is that only specific file formats are supported. For example, the only image files supported are JPEG, GIF, and BMP. Because nearly all image files are in one of these formats anyway, this restriction does not cause many problems. However, all three-dimensional imports must be VRML files. Because there are other three-dimensional modelling languages, this restriction severely limits the utility of DDS. Right now, users are required to translate any non-VRML objects into VRML on their own, but in the future, tools should be installed in DDS that can translate certain formats into VRML code. These tools do not need to be programmed from scratch; software to perform such translations already exist. If such software can not be directly incorporated into DDS, then at least a reference should be listed so that users can easily find the software to make the translation. A list of such software can be found at *http://www.sdsc.edu/vrml/translators.html.*

One more problem concerning imports deals with URL addresses: Currently, only absolute URLs are allowed. Presumably, users will often be downloading from their own machine; so relative URLs should also be allowed. Additionally, users should be allowed to download files onto the server side, because many clients will not have the files they want to import saved on a server machine.

Some other unimplemented features concern manipulation of imports and comments. Although objects can be moved around freely within the discussion space, they can not be rotated. In future versions of DDS, such a feature should be added. Another useful feature would be to allow resizing of comments and images. Currently, all comments use a predetermined font size, which determines the size of the comment window, and all images are set to the same size. Also, when a user imports something new, it would be easier for the user to move it to an appropriate location if the new object appears in front of the user. Currently, new imports pop up in front of the superdiscussion door; so if the user is not near the door, he may not know where to find it.

### VIII. Miscellaneous

Private chatting may prove useful for DDS users. Currently, if a user opens a chat window, his chat messages are sent to all clients currently logged in. Instead, he may want to just send messages to one specific client. Thus, DDS may add an option to direct messages at particular people. In that case, also having a feature to find out everyone who is logged in may also be useful.

Because Java 1.0 was the most recent version of Java available at the time DDS was started, the code is slightly outdated. The DDS code should be changed so that it meets the Java 1.1 specifications. Because Java's graphical interface package changed so dramatically between versions 1.0 and 1.1, this change may take a lot of time.

These are just a few of the many improvements that could be made upon the current version of DDS. Some can be made quickly, while others will take considerable thought and planning. The only way to find out what is truly missing from the application, though, is to release it to a number of users for beta testing.

## 5C. Future Uses of DDS

Eventually, DDS will hopefully expand to become a visual programming tool to create virtual environments besides discussions rooms. In other words, the user would have the option of enterring an environment in which three-dimensional objects and virtual worlds can be constructed, sort of like a Visual VRML.

In summary, Design Discussion Space was created to allow discussions among artists and designers in a three-dimensional environment over the Internet. However, DDS serves not only as a tool for artists and designers, but also for musicians, historians, and anyone else who wants to use it. DDS will greatly advance communications technology, and it may serve as a stepping stone to even more advanced online tools in the future.

## REFERENCES
## Literature

Bederson, B. and G. Furnas. "Space-Scale Diagrams: Understanding Multiscale Interfaces." *Proceedings of ACMSIGCHI.* ACM Press, 1995.

Bederson, B., J. Hollan, K. Perlin, and J. Meyer. Two Document Visualization "Techniques for Zoomable Interfaces." *Proceedings of ACM SIGCHI.* ACM Press, 1995.

Harold, Elliotte R. *Java Developer's Resource.* Upper Saddle River, NJ: Prentice Hall, 1997.

Hartman, Jed and Josie Wernecke. *The VRML 2.0 Handbook.* New York: Addison-Wesley, 1996.

Matsuba, Stephen N. and Bernie Roehl. *Using VRML.* Indianapolis: Que Corporation, 1996.

Perlin, K., "Coordination Theory and Collaboration Technology Workshop." *National Science Foundation*, June 1991.

Taber, Mark (publishing manager). *Java Unleashed.* Indianapolis: Sams.net Publishing, 1996.

## Web Pages

"3-D Website Builder." *http://www.virtus.com/products_3dwb.html.*

"Design Discussion Tool." *http://heinlein.media.mit.edu:8001/java/DDS/DDS.html.* (only accessible within MIT campus)

"Graphical Discussion Tool." *http://heinlein.media.mit.edu:8001/DiscussionHeinlein.html.* (only accessible within MIT campus)

"Java Packages". *http://java.sun.com/products/jdk/1.1.4/docs/api/.*

MacNeil, Ronald. "Civiscape: A Design Access Project." *http://civiscape.media.mit.edu/civiscape/.*

MacNeil, Ronald. "Civiscape Planning: future of GTT." *http://civiscape.media.mit.edu/civiscape/GTTplanning.html.*

"Onlive! Product Line." *http://www.onlive.com/prod/.*

"Onlive! Traveler Sites." *http://www.onlive.com/community/travelersites.html.*

“Package Index.”. *http://java.sun.com/products/jdk/1.0/docs/api/.*

"Stock Club and Onlive! Traveler." *http://stockclub.com/traveler.html.*

‘TerraVirtua Design.” *http://www.terravirtua.com.*

“TerraVirtua: Virtual Reality: Chat: Software.” *http://www.terravirtua.com/vr/vr_chat_software.html.*

“TerraVirtua: Virtual Reality: Software: Creation.” *http://www.terravirtua.com/vr/vr_creation_software.html.*

“VRML 2.0 specification.” *http://vrml.sgi.com/moving-worlds/spec/.*

“VRML Browsers.” *http://www.shareware95.com/netapps/browsers/browsing/vrml.htm.*

“VRML Repository: File Translators.” *http://www.sdsc.edu/vrml/translators.html.*

“VRML.SGI.COM.” *http://vrml.sgi.com.*

“WalkThrough Pro.” *http://www.virtus.com/products_wtp.html.*

# APPENDICES
## Appendix A. Instruction Manual

### I.. Logging In

When you first connect to DDS, a new window will appear on your desktop, as well as a VRML 2.0 browser in your web browser. (If you don't have a VRML 2.0 browser, you can download Cosmoplayer through *http://vrml.sgi.com.*)

Inside the new window, which we will call the prompt frame, a smaller window will appear, asking for your user name. If this is you first time logging in, simply choose a user name and hit OK (or enter). Next, a window should appear that prompts you to choose a password. If instead, a window appears asking for your password, then you have chosen a user name that has previously been used. In this case, hit cancel, and you will be reprompted for your user name. After enterring your password (twice, for confirmation), an account will be created for you, and you will be put in the home discussion, called "Main". You are now logged into DDS.

If you already have an account with DDS, enter your user name when prompted. Note that user names are case sensitive. After entering your user name, you should be asked for your password. If instead, you are asked to choose a new password, then you have probably misenterred your user name. In this case, hit cancel, and you will be prompted for your user name again. After enterring your password, you will be placed into the discussion from which you logged out. If that discussion is at capacity or has been removed since you last logged in, you will be placed in the home discussion.

### II. Importing objects

To import objects, go to the Imports menu in the prompt frame. Choose the appropriate menu item; then a window will appear in the prompt frame asking for the title and url of the import. After you supply this information and hit the "OK" button, the import is placed into the discussion in which you are currently standing. Note that no two objects within the same discussion may have the same title. After the object is added to the 3-D world, you should move the object to an appropriate place. (See Section IV: *Moving Objects*)

Supported image formats include JPEG, GIF, and bitmap, and anything else supported by VRML 2.0. Imported 3-D objects must be text files written in VRML 1.0 or 2.0.

## III. Posting Comments

From the Comments menu, choose "Post a Comment." A window will then appear in which you can type your comment. To start over, hit the "Clear" button. To post the comment, hit "Send". After you hit "Send," the comment will appear in the 3-D world. At this point, you should move the comment to the appropriate place (See Section IV, *Moving Objects*).

## IV. Moving Objects

To move a comment, image, or 3-D object in the 3-D world, click on the title bar of the object. A set of 3 axes will then appear. Choose an axis of motion by clicking on one of these axes. The chosen axis will light up. To move the object along that axis, click on the object, and drag. To move an object diagonally, two or three steps are required. For instance, if you want to move an object within the xy-plane, click on the x-axis, drag the object appropriately; then click on the y-axis, and drag the object again. When the object has been moved to the desired position, click on the title bar again. The axes will disappear, and the movement of the object will be reflected in the 3-D worlds of all users.

## V. Removing Objects or Comments

To remove comments, choose the appropriate option from the Comments menu. To remove images and 3-D objects, choose the appropriate option from the Imports menu. After choosing the appropriate option, a window will appear with a list of objects/comments that can be removed. This list will contain only items of the discussion you are currently in. Choose the appropriate item, and hit "OK."

## VI. Creating Discussions

To create a subdiscussion for the discussion you are currently in, choose "Create Subdiscussion" from the Discussions menu. You will then be asked for the title, description, and capacity of the discussion. The capacity means the maximum number of people that can be logged in to the subdiscussion at any given time. If someone tries to enter the subdiscussion when it is full, he will be told he can not enter because the subdiscussion is full. After hitting OK, a door should appear in the VRML world. This door corresponds to the new discussion room. (For more information on moving from room to room, see Section IX, *Navigation Between Discussion Rooms*.)

To create a new discussion room that is not a subdiscussion of any other discussion room, go to the home discussion by choosing "Main" from the Discussions menu. From there, choose "Create Subdiscussion," and follow the instructions above.

## VII. Removing Discussions

To remove a discussion, go to the superdiscussion of the discussion to be removed (see Section IX: *Navigation between Discussion Rooms*), and choose "Remove Subdiscussion" from the Discussion menu. You will then be shown a list of all the subdiscussions of the current discussion. Choose the appropriate title, then hit OK. The removal will be reflected in the VRML world.

Note that you can only remove a subdiscussion if it has no subdiscussions, and if no one is logged in to the subdiscussion. If you want to remove a tree of subdiscussions (i.e., a subdiscussion, all of its subdiscussions, all of their subdiscussions, etc.), you need to remove each room one at a time.

## VIII. Chatting

To open a chat window, choose "Chat" from the Comments menu. Any messages sent through this chat window will be sent to anyone logged in. When the sending chat window is created in your prompt frame, a receiving chat window appears in everyone else's prompt frame.

To send messages, click on the text area in the sending window, type your message, then hit "Send." When you hit "Send," the contents of the text area is displayed in all of the receiving chat windows.

## IX. Navigation Between Discussion Rooms

To move within the 3-D world, refer to the navigation instructions for the browser you are using. When you first enter a discussion room, its subdiscussions are to your left; the door to the superdiscussion is behind you; and the rest of the world is used for imports and comments. Referring to Figure 15, when you first enter, you are placed at the position marked "x", facing the navigation area.

The walls between the navigation area and the subdiscussions are transparent. From the navigation area, you see a bird's eye view of the subdiscussion, but the objects are not as detailed as they are when you enter the subdiscussion. Additionally, they are turned up so that you see the front view of the objects. In other words, when you view a subdiscussion from the navigation area, you see the bird's eye view of the layout of the discussion, but you see the objects as if you are looking at them from the front.

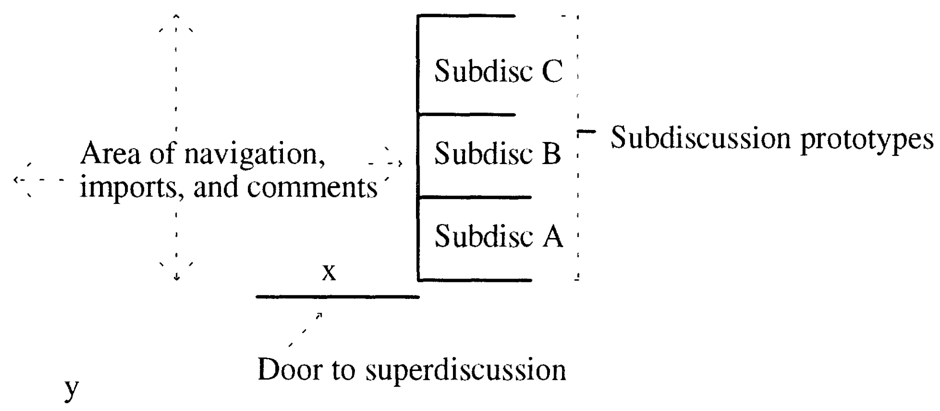


*Figure 15: General Layout of Discussion Spaces in DDS.*

To enter another discussion room, you can simply click on the door of the room that you want to enter. At this point, the contents of the new discussion room will appear, and you will be placed in the new discussion at position "x" in the diagram. Another way to enter other discussions is by using the prompting frame. The Discussions menu contains a list of all subdiscussions and the superdiscussion of the current room, as well as the home discussion ("Main"). If you click on any of these items, you will be moved to the chosen discussion room.

## X. History Windows

To display the history of a certain discussion, person, or both, choose "View history" from the File menu. After you choose this item, you will be shown two lists: One with all the names of the discussions in DDS, and one with all the users of DDS. You may choose a user, a discussion, or both. Once the choice is made, a history window appears in the prompt frame. If you chose a user, the history of the user is shown; likewise for discussions. If you chose a discussion and user, only the history of the chosen user within the chosen discussion is shown.

History windows contain records of every comment posted and chat sent, in chronological order. Each record also contains a time stamp of when the operation occurred. If any of these operations occur while the history window is open, it is updated appropriately.

## XI. Logging Out

To log out, go to the File menu and choose "Exit DDS." The prompting frame will be closed, and all objects will be removed from the 3-D world. The next time you log in, you will be placed in the same discussion from which you logged out. If the discussion is removed before your next log-in, or it is full the next time you log in, you will initially be placed in the home discussion.

## Appendix B. Class File List

The following files include one class, which has the same name as the name of the file, minus the ".java" extension:

| | | |
|---|---|---|
| ButtonBar.java | Comment.java | CommentProto.java |
| CommentTitleBar.java | ConfFrame.java | Database.java |
| Desktop.java | DiscChoice.java | Discussion.java |
| Display.java | EmbeddedCenteredWindow.java | |
| EmbeddedLeftJustifiedWindow.java | | EmbeddedWindow.java |
| ErrorWindow.java | GraphicFont.java | GraphicFontTextArea.java |
| Hashtable2.java | HistWindow.java | ImageLoader.java |
| Import.java | ImportInfo.java | ImportProto.java |
| InputLoop.java | LabelSet.java | ObjInfo.java |
| ObjMover.java | Person.java | Picture.java |
| PictureProto.java | Recorder.java | ScaleConstraint.java |
| ScaleLayout.java | ServerConnection.java | StreamManager.java |
| StringCode.java | Subdiscussion.java | SuperDisc.java |
| TalkSlot.java | ThreeDObj.java | ThreeDProto.java |
| TitleBar.java | TouchTranslator.java | WinDefault.java |
| WorldHandler.java | | |

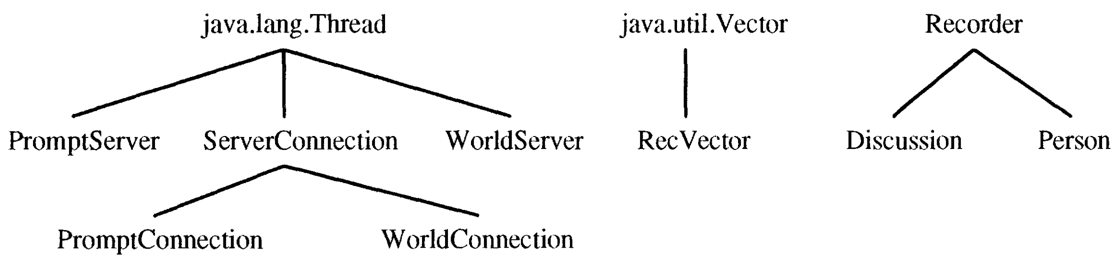
The following files have multiple classes. The classes are listed after the file name:

ConfClient.java -- ConfClient, WorldInLoop
ConferenceServer.java -- ConferenceServer, WorldServer, WorldConnection, WSInLoop
DisplayWindow.java -- Displaywindow, SelfSizedWin
PromptFrame.java -- PromptFrame, PromptInLoop
PromptServer.java -- PromptServer, PSInLoop, PromptConnection
PromptWindow.java -- PromptWindow, NameWindow, NewPwdwindow, AskPwdWindow, NewPicWin, RemovePrompt, NewDiscWin, Prompter
SendWindow.java -- SendWindow, SendCommWindow, SendChatWindow
TalkWindow.java -- TalkWindow, ChatWindow

## Appendix C. Class Hierarchy

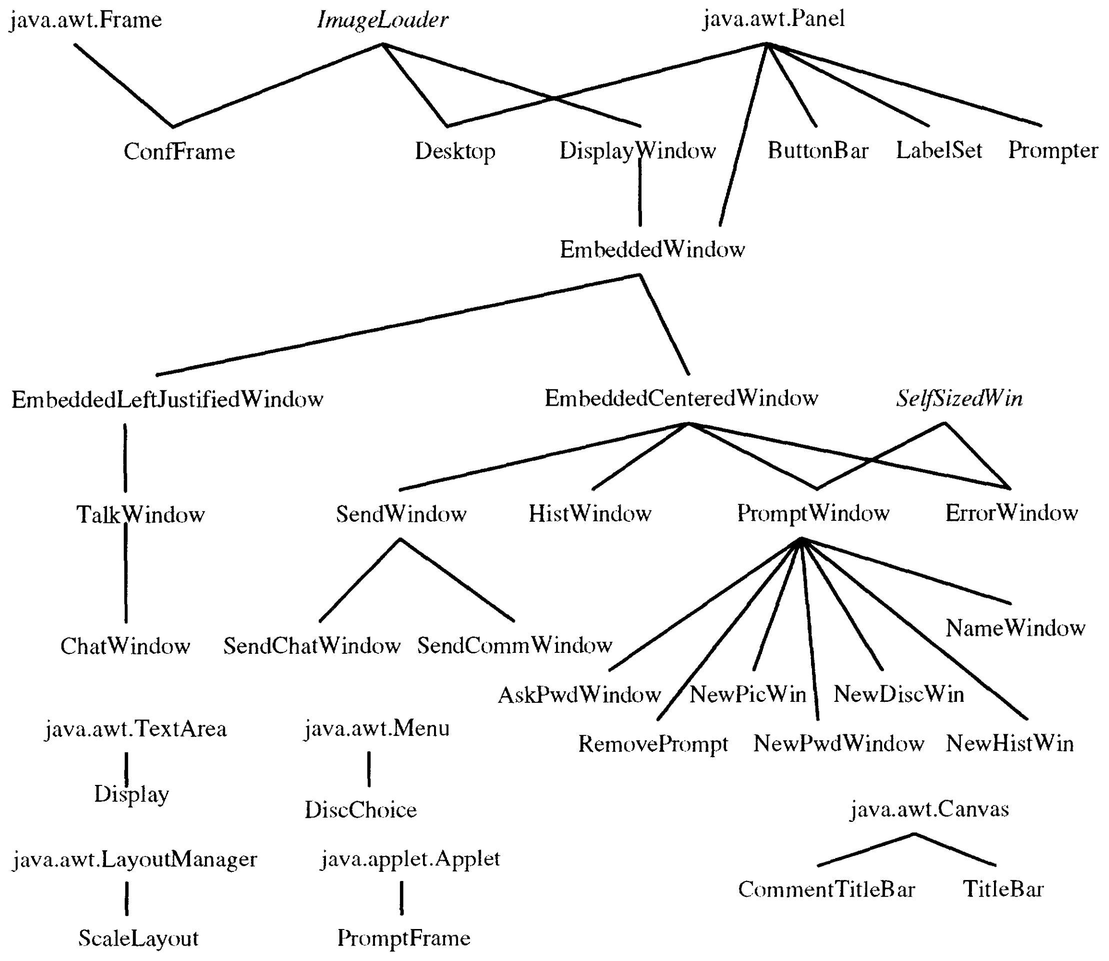
In all Diagrams, interfaces are written in italics.

## I. Server



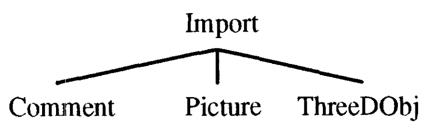
Other classes:
ConferenceServer Database ImportInfo Record
TalkSlot

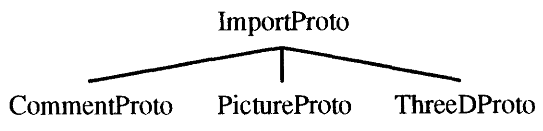
## II. Prompting Frame

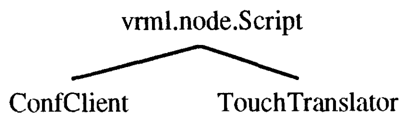


Other classes:
PromptInLoop ScaleConstraint WinDefault

## III. VRML World Frame



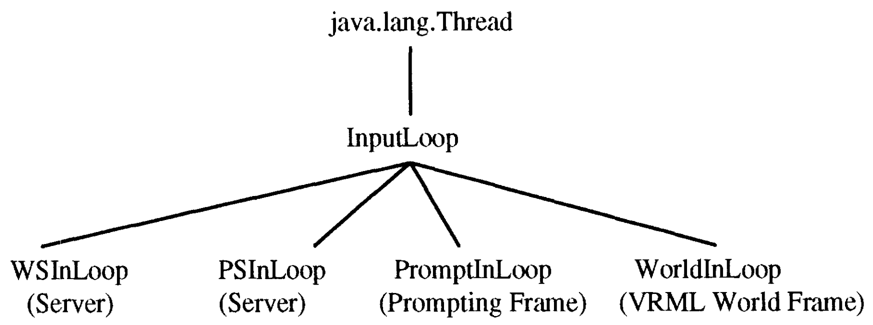




Other classes:

| | | | |
|---|---|---|---|
| ObjInfo | ObjMover | Subdiscussion | SuperDisc |
| WorldHandler | WorldInLoop | | |

## IV. Shared Classes



Other classes:

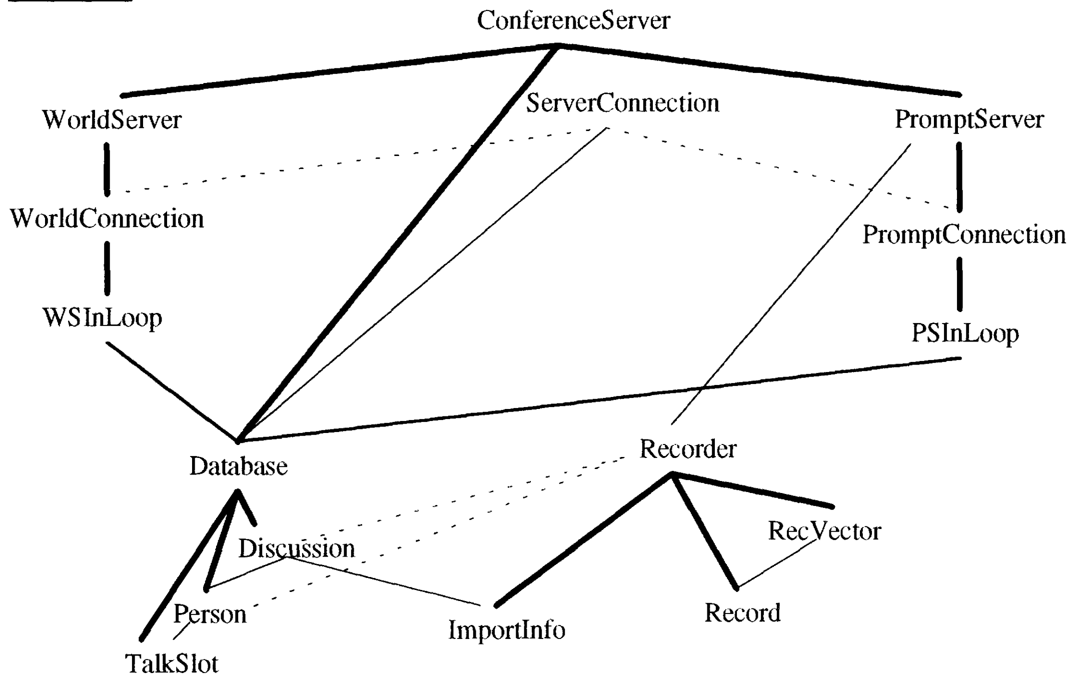
| | |
|---|---|
| StreamManager | StringCode |

## **Appendix D. Class Dependencies**

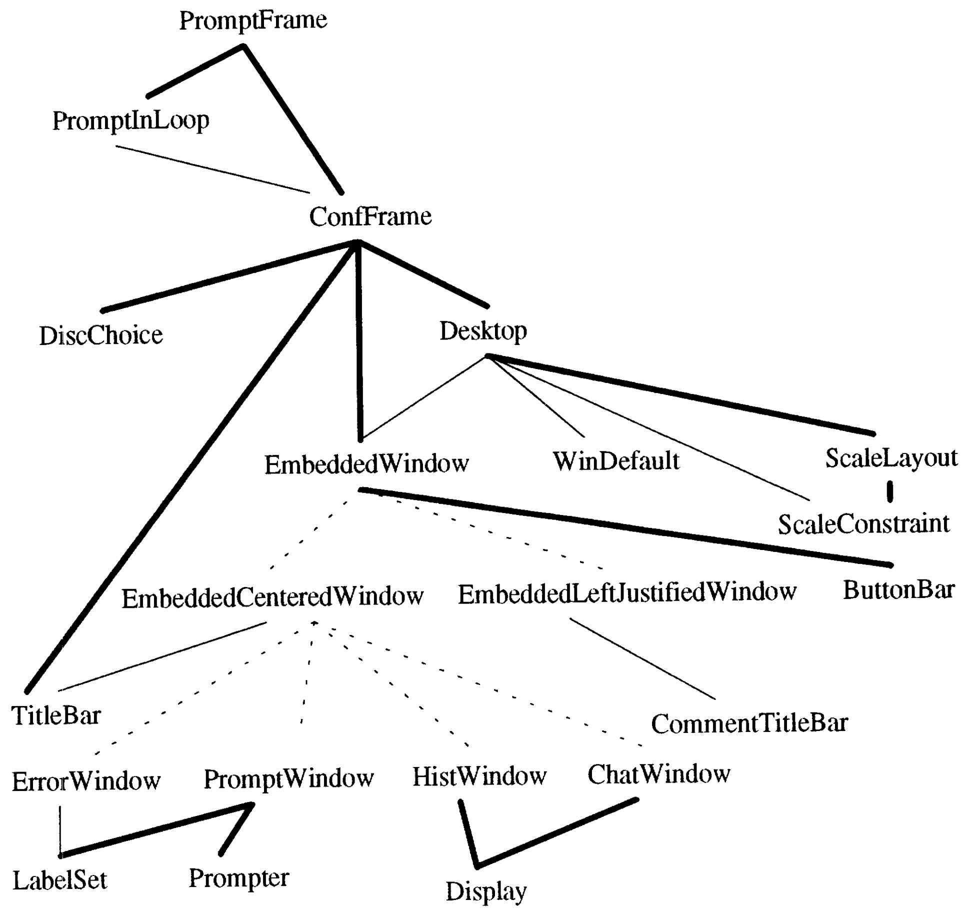
### I. Overview

In the following diagrams, dotted lines indicate parent/child relationships. Thick lines mean that the higher class creates the lower class. Thin lines mean that the higher class merely calls procedures in the lower class. If a class is dependent on a class X, it is dependent on all children of X; however these transitive lines are not drawn to prevent confusion. Interfaces are not included in these diagrams. Classes that add no new lines except for parent/child relationships have been omitted. The complete set of parent-child relationships is listed in the Appendix C, *Class Hierarchy.*

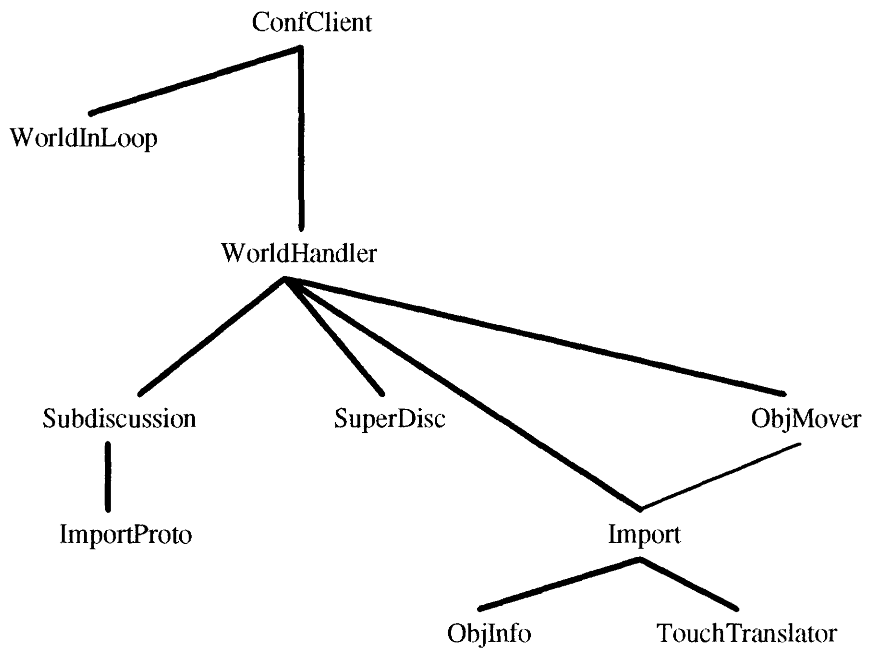
## II. Server


ConferenceServer
WorldServer
ServerConnection
PromptServer
WorldConnection
PromptConnection
WSInLoop
PSInLoop
Database
Recorder
Discussion
RecVector
Person
ImportInfo
Record
TalkSlot

## III. Prompting Frame



## IV. VRML World Frame



Note that Import creates TouchTranslator indirectly because it creates a node in the VRML world that in turn creates a TouchTranslator object.

## V. Shared Classes

StringCode and StreamManager are used by nearly every class. Because InputLoop is abstract, it is never called directly except by its children (PSInLoop, WSInLoop, PromptInLoop and WorldInLoop).

## VI. Network Communication

Because DDS is a server-client application, there is some dependency by means of communication over the network. The setup is rather simple: Message receiving is taken care of by an InputLoop, and message sending is taken care of by a StreamManager. The prompting frame receives messages from the server through the PromptInLoop class. WorldInLoop recieves the messages from the server for the VRML world frame. The server receives messages from the prompting frame through PSInLoop, and from the VRML world frame through WSInLoop.

For more information on communication, see Appendix E (*Message Protocol*) or the code files *StringCode.java* and *InputLoop.java* in Appendix H (*Code*).

## Appendix E. Message Protocol

### I. Overview

In the following explanation, all words in ALL CAPS represent String constants defined in the class StringCode (with the exception of START and STOP, which are defined in InputLoop). Each message sent from the server to the client, or vice versa begins with a message code, as defined by the constants below (in the Java code). If other information needs to be included in the message, the message code is followed by MESSAGEDIVIDER. Additionally, the main components of the message (i.e., the parameters) are separated by MESSAGEDIVIDER. If any of the parameters also contain subcomponents (for example, the coordinates of an object must contain 3 numbers), a different divider is used.

All messages begin with a START byte, and a STOP byte, as defined in InputLoop.java. As InputLoop reads in a message, it uses these bytes to distinguish one message from another.

IDDIVIDER is used to separate:

1) ordered triples, which are used for positions and sizes of objects

2) In identifying comments, to distinguish between each comment made by the same speaker, each comment is given an id number. IDDIVIDERs are used to separate speaker & id number when they are used to identify an object.

PARAMDIVIDER is used to separate:

1) in a layout file, an import identifier and its position. See Appendix G (*Layout File Format*) for more information.

On the server side, messages are received from the client's prompt frame through PSInLoop (see PromptServer.java). Messages are received from the VRML World through WSInLoop (see ConferenceServer.java). On the client side, messages are received by the prompt frame through PromptInLoop (see PromptFrame.java), and messages are received by the VRML world through WorldInLoop (see ConfClient.java).

In the following lists of code, the message code is followed by its list of parameters, with each parameter separated by a comma.

### II. Codes from Server to Prompting Frame

1) PERSONCAPSTRING (no parameters) -- This code is sent when all of DDS is at capacity. When this message is received, the user is told to try logging in at another time.

2) GETNAMESTRING, discussion list, person list -- This code means that the prompting frame should ask the user for his user name. The discussion list and person list is a list of

all discussions and users of DDS. This list is kept track of so that when a user wants to display a history window, he can choose from a list of discussions and/or people for which to display the history. When a user first connects to the server, either PERSONCAPSTRING or GETNAMESTRING is sent. After the user enters a user name, the prompt frame sends USERNAMESTRING to the server.

3) NEWNAMESTRING (no parameters) -- This code means that the user name the previously enterred does not match any user name registered with DDS. When this message is received, the prompt frame asks the user to enter a password, and gives the user the option of reentering his user name.

4) DISCNAMESTRING (no parameters) -- This code means that the user name previously enterred matches the name of a discussion that has been created. Because history files are for both discussions and people are saved under the name of the discussion or person, plus ".history", allowing a person and a discussion to have the same name would result in conflicts in the history file. When this message is received, the user is asked to enter another user name.

5) ACTIVENAMESTRING (no parameters) -- This code means that the user name previously enterred matches the name of a user currently logged in. When this message is received, the prompt frame asks the user to enter a user name again.

6) GETPWDSTRING, password -- This code means that the user name previously enterred matches a registered user name. When this message is received, the prompt frame asks the user for his password. If the password enters matches the password given, CREATEPERSTRING is sent to the server. Otherwise, the user is asked again to enter a password. The user also has the option of reentering his user name.

7) NEWUSERSTRING, user name -- This code means that another user with the given user name has logged in for the first time. This user name is added to the user list, which is displayed when a user requests to see a HistWindow.

8) NEWWINSTRING, window type, window parameters -- This code tells the prompt frame to create a window of the given type. Note that the "window parameters" mentioned are separated by MESSAGEDIVIDER. The number of window parameters vary depending on window type. Valid window types and their parameters are as follows:

a) CHATSTRING,speaker,initial text -- A chat window. Someone else logged in ("speaker") has created a SendChatWindow. The text of the newly created window is set to that indicated by "initial text."

b) HISTWINSTRING,discussion,speaker,initial text -- A history window. The user has requested a history window to be displayed for the given discussion and speaker. Note that either the discussion or speaker may be EMPTYFIELD, which is similar to a wildcard. For example, if discussion = EMPTYFIELD and speaker = "Bob", then the history window displays Bob's history over all discussions. The "initial text" is the initial text displayed in the history window. Strings may be appended to the display area as more comments are made.

c) ERRWINSTRING,title,message -- An error window. The message is displayed in a window with the title given. If the message is more than one line long, its lines are separated by MESSAGEDIVIDERS.

9) COMMENTLIST, list -- Sent when the user has just enterred a discussion. The list is a list of comment id's separated by PARAMDIVIDERs. Each comment id is the comment's speaker and id number, separated by and IDDIVIDER. When this message is received, the the list is parsed, and a comment list is created. This list is used when the user chooses to remove a comment, in which case the list of comments is displayed.

10) COMMENTSTRING, discussion, id, comment -- Someone has posted the given comment in the given discussion. The id is the speaker and this comment's id number, separated by IDDIVIDER. When this code is received, any relevant HistWindows are updated on the desktop. Additionally, the list of comments is updated if the comment was posted in the user's current discussion.

11) PICWINSTRING, title, url -- Someone has imported an image into the user's current discussion with the given title and url, or the user has just enterred a discussion. This message is sent to the prompt frame so that the list of imports can be updated. The list of imports is kept track of so that the list can be displayed when the user chooses a "remove" menu option. When a user first enters a discussion, he receives a series of PICWINSTRING and OBJSTRING messages to update his import list.

12) OBJSTRING, title, url -- Same as PICWINSTRING, but for imported VRML objects.

13) REMOVEIMPORT, discussion, type, id -- An import (comment/image/VRML object) has been removed from the given discussion, or a discussion his been removed from DDS. This code is misnamed because it used to only be used for imports. The type is PICWINSTRING, OBJSTRING, COMMENTSTRING, or REMOVEDISC. For images, VRML objects, and discussions, the id is the title. For comments, the id is the speaker and the comment's id number, separated by IDDIVIDER. For discussion removals,when this message is received, two checks are made:

a) Whether there is a user in the discussion to be removed
b) Whether is a user in one of the subdiscussions of the discussion to be removed

If either of these conditions are true, a NEWWINSTRING, with type ERRWINSTRING, is sent, to tell the user the situation. Otherwise, REMOVEDISC is sent to all client's promptframes, and to the VRML worlds of the clients who have a subdiscussion prototype displayed for the removed discussion. For imports, when the message is received, the appropriate element is removed from the appropriate import list if the given discussion matches the user's current discussion. Regardless, the appropriate history windows are updated.

14) SETTEXT, speaker, discussion, text -- The given speaker is in the given discussion and has sent a message (with the given text) through his SendChatWindow. The user's and history windows are updated appropriately.

15) CLOSECHAT, speaker -- The given speaker has closed his SendChatWindow. The corresponding ChatWindow on the current user's desktop is closed.

16) ADDSUBDISC, parent discussion, name of new discussion -- This message has a dual functionality: To add discussions to the discussion list, and to add discussions to the menu bar. If the new discussion is not already in the discussion list, it is added. This discussion list is used when a user wants to display a history window -- he chooses the discussion and person whose history he wants to view. Also, if the parent discussion is the same as the user's current discussion, the new discussion is added to the discussion menu. This message is used in 2 situations:

a) When the user has just enterred a discussion. In this case, DDS uses a series of ADDSUBDISC messages to create the Discussion menu.
b) When someone has created a new discussion. In this case, everyone currently logged in adds the discussion to his discussion list, and the people currently in the parent discussion add it to their Discussion menu.

17) ADDSUPERDISC, parent discussion -- Means that the user has just enterred a discussion whose parent discussion is as given. When this message is received, the name of the parent discussion is added to the Discussion menu. With the discussion menu, one can enter the parent discussion, subdiscussions, or the "home" discussion room (the room that has no superdiscussion).

18) CLEARDESKTOP, discussion -- Means that the user has just entered the given discussion. When this message is received, the Discussion menu is reset, as well as the import and comment lists. However, note that the discussion list is preserved, as well as the user list and any windows that are currently open on the Desktop. The name of the

message is slightly misleading, but the when the name was the message code was originally created for GTT, the whole desktop was actually cleared (i.e., all windows were removed.)

19) DISCFULLSTRING (no parameters) -- Sent when the user tries to enter a discussion, but that discussion is at its current capacity. When this message is received, an ErrorWindow is displayed on the Desktop explaining the situation. Whenever the user tries to change discussions, either CLEARDESKTOP or DISCFULLSTRING is sent from the server.

20) MAXDISCSTRING (no parameters) -- Sent when the user tries to create a new discussion, but DDS is at its capacity for number of allowed discussions. When this message is received, an ErrorWindow is displayed on the Desktop explaining the situation. If this is the case, a discussion needs to be removed in order for another discussion to be created.

## III. Codes from Server to VRML World Frame

1) COMMENTSTRING, comment id, comment text, coordinates -- Means that a comment with the given id and text should be added to the VRML world at the given coordinates. The id is the speaker and the comment id number, separated by IDDIVIDER. Each line of text is separated by a newline character ('\n'). The coordinates may have one of three types of values:

a) A set of 3 numbers, separated by IDDIVIDERS -- The coordinates are relative to the position 4 meters in front of the position when one first enters a discussion, which is 5 meters in front of the back wall. The positive x direction from the starting position is right; the positive y direction is up, and the positive z direction is backwards.

b) EMPTYFIELD -- This coordinate code is used when an object is first added to a world. (i.e., when someone chooses to import something or post a comment, but NOT when a user is first enterring a discussion room.) When this message is received, the object is placed at the default position.

c) REPLY -- This coordinate code is also used when an object is first added to a world. It also causes the object to be placed at the default postion, but also, the world responds to the server with a WORLDSIZE message, which tells the server the size of the world with this added object. REPLY is sent only to the world of the user who added the object; all other users receive EMPTYFIELD. Thus, only one response is received for the server's request for the new size of the world.

The imported Comment is a Billboard, which means that no matter where the user moves, the image will face him. When a user first enters the discussion, a series of COMMENTSTRINGs (as well as OBJSTRINGs and PICWINSTRINGs) are sent so that the user's world contains all the objects that have been imported into the world. Also, when a user in the user's current discussion room posts a new comment, COMMENTSTRING is sent.

2) PICWINSTRING, title, url, coordinates -- Means that an image with the given title and url should be added to the world at the given coordinates. The url should refer to a JPEG or GIF file (or any other types of files supported by VRML 2.0). The coordinates are in the same format as for COMMENTSTRING. The imported Picture is a Billboard, as explained above. A series of PICWINSTRINGs is sent when a user first enters a discussion. PICWINSTRINGs are also sent when someone in the user's current discussion imports a new image.

3) OBJSTRING, title, url, coordinates -- Same as PICWINSTRING, but for VRML objects. Also, ThreeDObjs are not Billboards.

4) OBJMOVEDSTRING, type, id, coordinates -- Sent when a DIFFERENT user in the same discussion room has moved an object. The type is COMMENTSTRING, PICWINSTRING, or OBJSTRING. For comments, the id is the speaker & comment id number, separated by IDDIVIDER. For images and VRML objects, the id is the title of the import. The coordinates are in the same format described above. They represent the new position of the coordinates. When the current user moves an object, OBJMOVEDSTRING is NOT received. Instead, the appropriate changes are made locally, and a message is sent when the object is moved. This message from the mover of the object (which also uses the OBJMOVEDSTRING message code, see below for protocol entry) triggers a series of OBJMOVEDSTRINGs to be sent from the server.

5) REMOVEIMPORT, import type, identifier -- Removes the import of the given type (which is COMMENTSTRING, PICWINSTRING, or OBJSTRING). The identifier is the same as defined above. This message is sent when a user has removed an object via the prompt frame. However, it is NOT sent when a user exits a discussion room -- in that case, CLEARDESKTOP is used.

6) ASKFORSIZE (no parameters) -- The server has requested the World to calculate its size. This message is sent after an object has been removed from the world. It is sent to only the world whose user removed the object (all other users currently logged in receive the OBJMOVEDSTRING); therefore, no multiple responses will be received. When the message is received, the world calculates its size by finding the most extreme coordinates of all objects (i.e., most negative x,y,z occupied by any object, and most positive x,y,z. These x, y, & z's are not necessarily all coordinates of the same object.) The world

responds by sending a WORLDSIZE. ASKFORSIZE is not needed when adding or moving objects for the following reasons:

a) When adding objects, the REPLY field can be sent as a parameter in place of the coordinates, which is equivalent to asking for the size.

b) When moving objects, the new size of the world is included in the OBJMOVEDSTRING sent from the client (see protocol entry below).

7) ADDSUBDISC, title, low bounds, high bounds -- Adds a subdiscussion prototype with the given title and bounds. The subdiscussion protype is simply a room with two side walls, no back wall, and a semi-transparent front wall with a semi-transparent door. The title is placed on the door to the prototype. The low bounds indicate the most negative (or least positive) coordinates of any part of any object in the subdiscussion. The high bounds indicate the most positive. From these coordinates, the size of the room is calculated, which in turn determines the shape of the subdiscussion prototype. Note that this message code only causes the walls and door of the prototype to be created; not the objects within the subdiscussion. Those objects are added using the ADDPROTO message code (see below). This message is sent when a user in the current discussion room creates a subdiscussion, or when the user first enters a discussion room that has subdiscussions.

8) ADDSUPERDISC, title -- Adds a superdiscussion prototype with the given title. The superdiscussion prototype is simply a door placed on the back wall. The title is written on the door. This message is sent when a user first enters a room that has a parent discussion. (All rooms will have parent discussions except for the "home" room.)

9) ADDPROTO, import type, subdiscussion topic, url, coordinates -- Adds an object of the given type with the given url to the prototype prototype for the given subdiscussion at the given coordinates. Type is COMMENTSTRING, PICWINSTRING, or OBJSTRING. Subdiscussion topic will be a subdiscussion that has been previously added by an ADDSUBDISC message. Url is the for pictures and VRML objects; for comments, this field is ignored (and is sent with the value EMPTYFIELD). Coordinates are an ordered triple separated by IDDIVIDERs. These coordinates are the same coordinates that would be given if the object were an object of the current discussion; they are transformed, rotated and scaled appropriately so that they are placed into the subdiscussion prototype.

10) CLEARDESKTOP, new discussion -- Sent when a user enters a new discussion. The name of the new discussion is the parameter given. When this message is received, the objects and subdiscussion prototypes, as well as the superdiscussion prototype, are removed from the world. This message is sent when the user first enters a discussion:

First, CLEARDESKTOP is sent; then a series of COMMENTSTRINGs, PICWINSTRINGs, etc. are sent to place all of the objects into the world.

## IV. Codes from Prompting Frame to Server

1) NEWUSERSTRING -- sent when the user first enters the page containing the PromptFrame applet. When this message is received, the server checks whether there is space in DDS for another person to enter. If so, it sends GETNAMESTRING. Otherwise, it sends PERSONCAPSTRING.

2) USERNAMESTRING, username -- sent when the user logs in. This message is sent in response to a GETNAMESTRING. If the user name given does not match any user names that have previously been used, NEWNAMESTRING is sent. If the name given matches a previous user name, and a person with the same user name is currently in DDS, ACTIVENAMESTRING is sent. Otherwise, a password is asked for by sending a GETPWDSTRING. In any case, the username sent is recorded for future use. In other words, this PSInLoop now knows which user it is communicating with.

3) CREATEPERSTRING, password -- sent when the user has enterred a correct password, in response to a GETPWDSTRING, or when the user has created a new password (thereby creating a new account), in response to NEWNAMESTRING. When this message is received, a new person is enterred into the database. If the username is new, a new entry is created in the passwords list.

4) NEWDISCSTRING, topic, description, capacity -- The user has created a subdiscussion in his current discussion with the given capacity, topic, and description. When this message is received, a new Discussion class is created, an ADDSUBDISC is broadcasted to all prompt frames of users who are logged in, and an ADDSUBDISC is sent to all worlds of users who are currently in the parent discussion.

5) CHANGEDISC, new discussion -- The user has change to the given discussion. When this happens, the person is removed from his current discussion and added to another, which means that a CLEARDESKTOP is sent to the prompt frame and the world, followed by a series of COMMENTSTRINGs, a COMMENTLIST, PICWINSTRINGs, OBJSTRINGs, ADDSUBDISCs, an ADDSUPERDISC, and ADDPROTOs. However, if the discussion to which the user wants to change is already at capacity, a DISCFULLSTRING is sent.

6) PICWINSTRING, title, url -- User has requested to import an image with the given url and title in his current discussion. When this message is received, a PICWINSTRING message is sent out to every person currently in the same discussion, new records are created in the history files, and an ASKFORSIZE is sent to the importer.

7) OBJSTRING, discussion, title, url -- This does exactly the same thing as PICWINSTRING, but for 3-D objects.

8) COMMENTSTRING, comment -- The user posted a comment in his current discussion. When this message is received, a COMMENTSTRING is sent out to all promptframes, and the worlds of users currently in the same discussion room.

9) REMOVEIMPORT, import type, import id -- The user wants to remove the given import. The import type is COMMENTSTRING, PICWINSTRING, or OBJSTRING, and the id has the same format as the id's described above (see protocol entry for OBJMOVEDSTRING in list of messages from server to VRML world) When this message is received, removal records are added to the history files, and REMOVEIMPORT messages are sent to all VRML worlds of people in the same discussion (including to the person who removed the import) Additionally, an ASKFORSIZE message is sent to the user who removed the import.

10) CHATSTRING (no parameters) -- The user has opened a SendChatWindow. When this message is received, a NEWWINSTRING is sent so that a new ChatWindow is opened in everyone's prompt frame, except for that of the person who opened the SendChatWindow. Note that if a user tries to open two SendChatWindows, a CHATSTRING message is not sent for the second window. In other words, all messages from both SendChatWindows are displayed in the same ChatWindow on other peoples desktops.

11) CLOSECHAT (no parameters) -- The user has closed his SendChatWindow. When this message is received, a CLOSECHAT is sent out to everyone except for the user who was chatting. Note that if the user had two SendChatWindows open, CLOSECHAT is not sent; so CLOSECHAT is only sent from the prompt frame when the last SendChatWindow is closed.

12) HISTWINSTRING, discussion, speaker -- The user wants to open a history window for the given discussion and speaker. When this message is received, the history of the appropriate discussion & speaker are retrieved, and a NEWWINSTRING, with a HISTWINSTRING parameter, are sent to the user.

13) SETTEXT, text -- The user has sent a message through his SendChatWindow. When this message is received, the history files of the appropriate discussion & speaker are updated, and the message is sent to everyone logged in except for the person sending the message.

14) DONESTRING -- The user has exited DDS. Connections are closed with the client. This message is sent either when the user hits the Exit button, or when he closes his

browser.

V. Codes from VRML World Frame to Server

1) NEWUSERSTRING -- This message is sent when the user first enters the page (before he logs in). When this message is received, a new connection with the VRML world is created.

2) OBJMOVEDSTRING, object type, identifiers, new position, minimum dimensions of world, maximum dimensions -- This message is sent when the user moves an object in the VRML world. Note that it is only sent once the user clicks on the title bar to set the object in place, not while the object is actually moving. the minimum/maximum dimensions of the world are the smallest and largest x, y, and z coordinates of the world after the object is moved, respectively.

3) COMMENTSTRING, comment id, position -- This message is sent when a comment is added to the VRML world. It is only sent in response to a COMMENTSTRING sent from the server to VRML world, if the position parameter of the server's message was REPLY. When this message is received, the layout file is updated with the new comment. This roundabout way of finding the initial position of the object is used so that the client side is always responsible for "knowing" the position of the objects, and where they should be placed. Thus, the server does not know where the initial position of the object will be, until it asks the client. Only one client is asked to reply.

4) PICWINSTRING, title, position -- This message is sent when an image is added to the VRML world. It works exactly like COMMENTSTRING (see above).

5) OBJSTRING, title, position -- Once again, this message works just like COMMENTSTRING, but for 3-D imports.

6) CHANGEDISC, new discussion -- The user has clicked on a door to another discussion. This message is handled just like a CHANGEDISC from the promptframe (see protocol entry above.)

7) WORLDSIZE, min dims, max dims -- This message is sent in response to either an ASKFORSIZE from the server, or when an object is added (via COMMENTSTRING, PICWINSTRING, or OBJSTRING), and a REPLY is in the position field. The dimensions given are the minimum and maximum values of x, y, and z that are occupied by any object in the discussion. When this message is received,the layout file for the appropriate discussion are updated

## Appendix F. History File Format

The names of the history files of users are the same as the user name, with ".history" appended to the end. For example, the history of user "Danai" would be saved in the file "Danai.history." The names of history files of discussions are the topic of the discussion, with ".history" appended to the end.

Below are the formats of every record type for history files. Each record of a history file is divided by an untypable line of characters. The constant for this divider is defined in StringCode as HISTDIVIDER. Note that the first line refers to a String constant in StringCode. (see StringCode.java in Appendix H) When there is more than one String constant listed on the first line, only one of them is used. Each String constant listed is respective to the record types listed above the format. For example, Comments always start with the line StringCode.COMMENTSTRING, which is equivalent to "Comm".

Comments and Chats:
COMMENTSTRING/CHATSTRING
[user name of speaker]
[title of discussion room]
[timestamp]
[content (what was actually said)]

Imports of Images and 3-D Objects:
PICWINSTRING/OBJSTRING
[user name of importer]
[title of discussion into which object was imported]
[timestamp]
[title of import]
[url for import]

Subdiscussions:
NDSC
[user name of creator]
[title of new discussion room]
[title of parent of new discussion room]
[timestamp]
[maximum capacity of new room]
[description of new room]

Removed comments/imports:

```
RIMP
[user name of remover]
[discussion from which object was removed]
[timestamp]
[title of import/id of comment]
```

id of comment means the speaker of the comment, followed by StringCode.IDDIVIDER, followed by the comment id number. Every comment for a particular user has a distinct id number.

History files of users end with StringCode.DONESTRING, followed by the name of the discussion from which the user last logged out.

Here are some short examples of history files:

```
NDSC
Danai
subdisc
Main
Tue May 13 23:53:38 EDT 1997
10

-•------------
Pwin
Danai
subdisc
Tue May 13 23:54:12 EDT 1997
raskin
http://campbell.media.mit.edu:8001/DDS-Models/raskin.jpg
-•------------
3Obj
Danai
subdisc
Wed May 14 20:16:43 EDT 1997
thingy
http://campbell.media.mit.edu:8001/DDS-Models/testObj1.wrl
-•------------
Comm
Danai~0
Main
Thu May 15 00:37:08 EDT 1997
Hello world
From the main discussion room
-•------------
```

```
DONE
subdisc
```

This file is the history of a use with username Danai. The “-•------------“is the divider between history records. The first record represents the creation of a new subdiscussion of the “Main.” This new discussion was named “subdisc”. The last line before the history divider is blank because the user did not provide a description for the new subdiscussion. The second and third records represent the importation of a two-dimensional image and a three-dimensional object into subdisc. The fourth record represents the posting of a comment in Main. And the fifth record is not really a record; it merely marks the end of the file, and says that when the user last logged out, he was in subdisc.

Here is the history file for the discussion called “Main”:

```
NDSC
Danai
subdisc
Main
Tue May 13 23:53:38 EDT 1997
10

-•------------
Comm
Danai~0
Main
Thu May 15 00:37:08 EDT 1997
Hello world
From the main discussion room
-•------------
```

Notice that these records are identical to the ones in Danai’s history file. Only the actions that were performed within Main are included, of course. Also, notice there is no record starting with “DONE” at the end of this file.

## Appendix G. Layout File Format

Each discussion has a separate layout file. The name of the layout file is the name of the discussion, with ".layout" appended to the end. DDS checks when a file is created to make sure that no other previously created discussions have the same name. Therefore, there is no chance that two discussions will be writing to the same layout file. In the following explanation, all words in ALL CAPS are String constants defined in *StringCode.java.* (See Appendix H, *Code*).

The first two lines of the layout file indicate the size of the discussion. Both lines are ordered triples, with each coordinate separated by an IDDIVIDER. The first line has the smallest or most negative x, y, and z coordinates occupied by any object in the discussion, while the second has the greatest or least negative coordinates. In other words, the two points are corners of the smallest bounding box for the discussion.

The next set of lines are the comments and imports of the discussion. Each object occupies one line. Each line has an identifier for the object, and the position of the center of the object The identifier and position size are separated by a PARAMDIVIDER. The identifier, for images and 3-D objects, is simply the title of the object. For comments, it is the person who posted the comment, followed by an IDDIVIDER, followed by an id number for the comment. Each comment for the same speaker has a distinct id number. ID numbers are assigned as follows: The first comment posted by a person has id number 0. The next one is 1; the next one is 2; and so forth. The id number continues to increase even if the user posts a comment in a different discussion. The position of the object is an ordered triple, with elements separated by an IDDIVIDER.

After the list of the objects is a DONESTRING. There may also be junk after the DONESTRING. This junk results when an object is removed from the world -- The object is removed from the list in the layout file, meaning that the DONESTRING is pushed up, but the part of the file that was originally at the end of the file is not removed.

Here are some example layout files:

```
-1.65~-0.35~-4.65
1.65~0.35~-1.35
Danai~0`0~0~-3
DONE
```

In this discussion room, there is only one comment, and no imports. The boundaries of the entire discussion room are indicated by the first two lines: In the x-direction, the room is bounded between -1.65 and 1.65; in the y-direction between -0.35 and 0.35, and in the z direction between -4.65 and 1.35. Note that the tilda ("~") is, in this case IDDIVIDER. The comment in this discussion room was posted by Danai; it was his first comment (indicated by the 0 after "Danai~"); and it is at position (0, 0, 3) in the discussion room. (So, " ` " is the PARAMDIVIDER) At the end of the file is StringCode.DONESTRING (= "DONE").

It is possible that someone names an import "Danai~0." In such a case, there would be confusion in the layout file, because the lines that record the position of the comment and of that import would both begin with "Danai~0". However, IDDIVIDER, as well as PARAMDIVIDER, is only temporarily a typable character (i.e., "~"), to facilitate debugging. In the final version of DDS, all dividers will be untypable.

Here is another, slightly more complicated layout file:

```
-7~-2.5~-4.5
1.5~2.5~-1.5
```

```
raskin`0~0~-3
thingy`-5.5~0~-3
DONE
E
```

Again, the first two lines represent the boundaries of the room. The second and third lines are imports whose titles are “raskin” and “thingy”, and positions are (0, 0, 3), and (-5.5, 0, 3), respectively. The “DONE” indicates the end of the list of imports. So, the “E” on the last line is ignored. This “E” probably resulted from someone removing an import from the room. Originally, that “E” was the last letter of a line that said “DONE”, but since an import was removed, the line containing the position of that import was removed, and the “DONE” was rewritten at an earlier position in the file. However, the length of the file does not shorten; so some of the old “DONE” still remains.

## Appendix H. Code

The following pages include all code files of DDS. Each file begins with a set of comments that includes the name of the file, what classes are included in it, and a brief description of each class.

### I. Server

```
/* ConferenceServer.java
Contains ConferenceServer, WorldServer, WorldConnection,
   and WSInLoop classes.

ConferenceServer is a multithreaded server that handles requests from
conference. It contains a WorldServer and a PromptServer, which are both
contacted when a client requests to enter the discussion (see
ConfClient.java). The WorldServer then opens a WorldConnection and
starts a new Thread, the WSInLoop. The WSInLoop takes
care of all messages from that particular client. PromptServer also opens
a new connection through which it sends broadcasted messages (see
PSInLoop.)
*/

import java.net.*;
import java.io.*;
import java.util.*;
import ServerConnection;
import InputLoop;
import StringCode;
import StreamManager;

class ConferenceServer {
```

```
// CONSTANTS
  public static final boolean DEBUG = true;

// FIELDS
//  private ConferenceManager conference;
  private static Database database;
  private static WorldServer ws;
  private static PromptServer b;
//  public Vector open_slots = new Vector(10, 2);


  public ConferenceServer(int broadcast_port, int request_port)
  //UsedNameException should never actually be thrown in this case, but because
  //        the Database constructor throws it sometimes, it must be included in
  //        throws clause
  {
//   if (DEBUG) System.out.println("broadcast_port:"+broadcast_port+" "+
//                          "request_port:"+request_port);
//   if (DEBUG) System.out.println("Done initializing conference manager!");
    b = new PromptServer(broadcast_port);
    database = new Database(StringCode.DEFAULTDISC,
           "Contains all other discussions",b,ws);
    ws = new WorldServer(request_port, database, b);
    b.start();
    ws.start();
  } // end ConferenceServer constructor

  public static void main(String args[])
  // effects:  reads broadcast_port and request_port from command line,
  // then creates ConferenceServer
  {
    new ConferenceServer(StringCode.PROMPT_PORT,StringCode.WORLD_PORT);
  } // end main

} // end ConferenceServer

/*********************************************************************/

// This guy waits for a connection then spawns a ServerConnection to deal with
// it
class WorldServer extends Thread {

//CONSTANTS
  private final static boolean DEBUG = ConferenceServer.DEBUG;

// FIELDS
  private static int port;
  private static Vector connections = new Vector(2,1);
//  private Vector outputs = new Vector (2,1);
```

```
  private ServerSocket main_socket;
  static Database database;
//  static PromptServer bc;

// METHODS

  public WorldServer(int _port, Database d, PromptServer b) {
    port = _port;
    database = d;
//    bc = b;
  } // end WorldServer

  public void run()
  {
    try {main_socket = new ServerSocket(port);}
    catch(Exception e) {StreamManager.showError(e);}

    WorldConnection temp_sc;
    while (true)
    {
      try
      {
            System.out.println("ConnectionManager:Waiting for connection");
        Socket this_connection = main_socket.accept();
        System.out.println("World Connection!");

        // spin off new thread to handle request if there's room
        // otherwise, a message will be sent through the broadcast
        // connection saying that DDS is full
        if (!database.atCapacity()) {
          temp_sc = new WorldConnection(this_connection, database, this);
          temp_sc.start();
          connections.addElement(temp_sc);
//          outputs.addElement (temp_sc.output);
        }

        //clean up the vector if needed
        for(int i=0;i<WorldServer.connections.size();i++)
          if(!((ServerConnection)(connections.elementAt(i))).isAlive())
            connections.removeElementAt(i);
      }
      catch(Exception e) {StreamManager.showError(e);}
      yield();
    }                   //while true
  } // end run

  public void broadcast(String s)
  // effects:  outputs s to all broadcast sockets
  {
    if (DEBUG) System.out.println ("broadcasting to promptframes: "+s);
//    StreamManager out;
    Enumeration enum = connections.elements();
    while (enum.hasMoreElements())
      ((StreamManager)((WorldConnection)enum.nextElement()).output).send(s);
  } // end broadcast (String)

  public void removeConn (WorldConnection c)
  //removes given connection from connections list
  {
    if (DEBUG) System.out.println ("removing worldconnection from list");
    if (!connections.removeElement(c))
      System.err.println ("Error: worldconnection not found in list");
  }


} // end WorldServer

/**********************************************************************/

// This class encapsulates one socket connection
class WorldConnection extends ServerConnection
{
//CONSTANTS
  public final static boolean DEBUG = ConferenceServer.DEBUG;


// FIELDS
//  static Database database;
//  static PromptServer bc;
  protected WSInLoop input_loop;
  private WorldServer ws;

// METHODS
  public WorldConnection(Socket s, Database d, WorldServer w)
  {
    super(s,d);
    System.out.println("WorldConnection:init");
    ws = w;
//    database = d;
//    bc = b;
    input_loop = new WSInLoop(this, d, w);
  } // end WorldConnection constructor*/

  protected void doServerWork()
//  throws InterruptedException
  // effects:  starts WSInLoop
  // returns the Exception thrown, if any.
  // this exception isn't rethrown because the compile won't let it.
  // if it tries to throw an exception, it says Invalid Exception thrown
  // but if it doesn't throw it, it says, Exception Must Be declared in
```

```
 //   throws clause
 {
  try {
    input_loop.start();
    input_loop.join();
//    input_loop.run(this);
   } catch (Exception e) {StreamManager.showError(e);}
  dprint ("done with WorldConnection");
 } // end doServerWork

 public WSInLoop getInputLoop() {return input_loop;}

 protected void handleSocketException ()
 //called if connection with client is lost
 //e.g., if user closes browser without hitting "Exit" button
 {
  dprint ("Handling socket exception in WorldServer");

  //if user hasn't logged in yet, simply close connection
  if (database.removeStream(this))
    dprint ("stream successfully removed");
    //removeStream is called just in case there is no PSInLoop.  Perhaps
    //the user is just viewing the WSInLoop now, and PSInLoop has been
    //terminated.  If there is a PSInLoop, it will also try to remove
    //this connection, but that's ok, because the resulting
    // NullPointerException is caught (see Database.removeStreams())
  else dprint ("one or more streams could not be found");

  //if he has, let PromptConnection take care of removing this connection
  //from the database
 }

      public void destroy()
      {
       System.out.println ("WorldConnection.destroy() called");
       ws.removeConn(this);
       try {
         mysocket.close();
       } catch (IOException e) {}
//          System.out.println ("IOException thrown");
//        }
//        input_loop.stop();             //done with this client
            stop();

//        super.destroy();
      }

  public void dprint (String s)
  {if (DEBUG) System.out.println (s);}
```

```
} // end WorldConnection

/*********************************************************************/

class WSInLoop extends InputLoop
// This class constantly checks the request socket to see if there is
// any input. If there is, then processes request accordingly.
// In DDS, the request socket is the socket that communicates with the
//    client's VRML world (managed by WorldHandler --
//    see WorldHandler.java)
{
//FIELDS
 WorldConnection conn;
 String name;  //of user
 static WorldServer ws;
 static Database database;
 int count = 0;    //used by displayRunning()

// METHODS
 public WSInLoop(WorldConnection c, Database d, WorldServer w) {
  super(c.input);
  ws = w;
  database = d;
  conn = c;
 } // end WSInLoop constructor

 public void set_name (String n) {name = n;}

 public boolean process_message(String client_request)
 // effects: what requests the server responds to are specified here.
 // to add new requests, simply add another "if" statement
 {
  if (DEBUG) System.out.println("processing (world):" + client_request);
  StringTokenizer tok = StreamManager.parseMessage (client_request);
  String code = tok.nextToken();
  if (DEBUG) System.out.println ("code: "+code);
  try {
   //FOR NEW USERS
   if (code.equals(sc.NEWUSERSTRING)) {
    //user just enterred page on browser
    //adds the request output to the database
    //this output will be paired with the broadcast output from
    //the same machine
    database.addWConn ((WorldConnection)conn);
    return true;
   }   //if code.equals NEWUSERSTRING

   //WINDOWS
   if (code.equals(sc.OBJMOVEDSTRING)) {
        //object in 3-D world was moved
        //notify other users
        database.moveObject (name,tok.nextToken(),    // object type
                tok.nextToken(),tok.nextToken()); //object identifiers, new pos
        return true;
  }

  //WHEN IMPORTS ARE ADDED
  if (code.equals(sc.COMMENTSTRING)) {
       //comment was added to 3-D world
       //record its position
       database.recordPos(sc.COMMENTSTRING,name,tok.nextToken(), //comment id
                                                   tok.nextToken()); //position
       return true;
  }               //if code.equals COMMENTSTRING

  if (code.equals(sc.PICWINSTRING)) {
       //comment was added to 3-D world
       //record its position
       database.recordPos(sc.PICWINSTRING,name,tok.nextToken(), //title
                                                   tok.nextToken()); //position
       return true;
  }               //if code.equals PICWINSTRINGG

  if (code.equals(sc.OBJSTRING)) {
       //comment was added to 3-D world
       //record its position
       database.recordPos(sc.OBJSTRING,name,tok.nextToken(), //title
                                           tok.nextToken()); //position
       return true;
  }               //if code.equals OBJSTRING

  //OTHER
  if (code.equals(sc.CHANGEDISC)) {
    //changes appropriate person to appropriate discussion
    database.movePerson (name,tok.nextToken());
    return true;
  } //if code.equals CHANGEDISC

  if (code.equals(sc.WORLDSIZE)) {
       //indicating size of discussion's corresponding VRML world
       database.setWorldSize (name,tok.nextToken(),tok.nextToken());
       return true;
  } //if code.equals WORLDSIZE

  System.err.println ("Error: invalid message received by server");
 } catch (Exception e) {StreamManager.showError(e);}
 System.out.println ("Message not processed: "+client_request);
 return false;
} // end process_message
```

```
 protected void handleSocketException()
 {
//   super.handleSocketException();
   conn.handleSocketException();
 }

 protected void displayRunning()
 //temporary
 //used to see whether input loops are hogging processor (when there is
 //        only one client logged in)
 {
//   if (count==0) System.out.println ("WSInLoop waiting for message");
//   else
         if (++count>10000) count = 0;
 }
} // end WSInLoop
```

---

```
/*Database.java
Contains only database class.

This class managers all database information.  It is the only class
contacted by PromptServer & WorldServer when messages come in from the
client.  It keeps track of all people, discussions, passwords, and
chat slots. (see Person.java and Discussion.java)
*/

//package DatabasePack;
import java.awt.*;
import java.util.*;
import java.io.*;
import java.net.InetAddress;
import StreamManager;
import StringCode;
import PromptServer;
import TalkSlot;
//import UsedNameException;

public class Database {
//Database is a Vector of discussions, with some added fields & functions
//CONSTANTS -- only to be used in DatabasePack
  public static final boolean DEBUG = true;
  public static final String PWDFILE = "passwords"; //file with user passwords
  public static final StringCode sc = new StringCode();
  public static final int MAXDISCSIZE = sc.MAXDISCSIZE;
//FIELDS
  private Vector DiscList=new Vector(3,1);  //List of discussions
  private Vector PersonList=new Vector(5,1); //People currently logged in
  private int MaxPeople=100;  //Maximum total number of people
  private int MaxDiscs=10;  //Maximum number of discussions
  private Hashtable Passwords = new Hashtable (10); //list of passwords of
```

```
        //everyone who has ever logged in
  private Vector SlotList = new Vector (2,1);
  //the following 2 hash tables used to identify which world and prompt
  //connections should be paired together.  They are paired if they come from
  //the same machine.
  private Hashtable unpairedWConns = new Hashtable (1);
  private Hashtable unpairedPConns = new Hashtable (1);
  public static PromptServer ps;
  public static WorldServer ws;

//CONSTRUCTORS
  public Database (String topic, String descript, PromptServer b,
              WorldServer w)
  {
   dprint ("creating Database");

   loadPasswords();  //loads passwords from file
   Discussion d;
   ps = b;
   ws = w;
   b.database = this;
//   try {
    d = new Discussion (topic, descript, ps, (Discussion)null,
                          MaxPeople);
    DiscList = d.getDescendants();
    DiscList.addElement(d);
/*   } catch (UsedNameException e) {
    System.err.println ("UsedNameException thrown in Database constructor");
    System.exit(1);
   }  //try/catch*/
   dprint ("DiscList: "+DiscList);
  }  //Database constructor

//PUBLIC FUNCTIONS
//get and set functions
  public String getDiscs()
  //returns string representation of all discussions
  //discussions are separated by sc.PARAMDIVIDER
  {
   dprint ("Converting DiscList to string");
/*
   StringBuffer returnstr = new StringBuffer();

   for (Enumeration enum = DiscList.elements(); enum.hasMoreElements();) {
    returnstr.append(((Discussion)enum.nextElement()).toString());
    if (enum.hasMoreElements()) returnstr.append (sc.PARAMDIVIDER);
   }
*/
   String returnstr = StreamManager.unparseList(DiscList.elements());
   dprint ("DiscList = "+returnstr);
  return returnstr;
 }

 public String getPeople()
 //returns string representation of all people ever logged in
 //each name is separated by sc.PARAMDIVIDER
 {
  dprint ("Converting list of people to string");

  String returnstr = StreamManager.unparseList (Passwords.keys());
   //here, Passwords.keys are the names of the people, but not
   //their passwords
/*
  for (Enumeration enum = Passwords.keys(); enum.hasMoreElements();) {
   returnstr.append((String)enum.nextElement());
   if (enum.hasMoreElements()) returnstr.append (sc.PARAMDIVIDER);
  }
*/

  dprint ("AllPeople = "+returnstr);
  return returnstr;
 } //getPeople()

 public int getMaxPeople() {return MaxPeople;}
 public int getMaxDiscs() {return MaxDiscs;}

 public int NumPeople () {return PersonList.size();}
 public int NumDiscs () {return DiscList.size();}

//COMMUNICATIONS METHODS
 public void addWConn (WorldConnection output)
 //adds output to list of WorldConnections waiting to be paired with
 //PromptConnections
 {
  dprint ("adding unpaired WConn for "+output.getInetAddress());
  unpairedWConns.put (output.getInetAddress(),output);
 }

 public void addPConn (PromptConnection output)
 //adds output to list of PromptConnections waiting to be paired with
 //WorldConnections
 {
  dprint ("adding unpaired PConn for "+output.getInetAddress());
  unpairedPConns.put (output,output.getInetAddress());
 }

 public boolean removeStreams (PromptConnection pconn)
 //removed and close unpaired streams corresponding to broadcast
 //connection bconn
 //returns true if removal is successful
 {
  try {
    InetAddress adr = pconn.getInetAddress();
    if (adr == null) return false;
    dprint ("removing unpaired streams for "+adr);
    WorldConnection wconn =
      (WorldConnection) unpairedWConns.get (adr);

    unpairedPConns.remove(pconn);
    unpairedWConns.remove(adr);
    wconn.destroy();     //close sockets
    pconn.destroy();
  } catch (NullPointerException e) {
    return false;
  } catch (Exception e) {
    StreamManager.showError(e);
    return false;
  } //try/catch
  return true;
 }    //removeStreams

 public boolean removeStream (WorldConnection conn)
 //removes conn from list of unpaired of WorldConnections
 //returns true if connection was removed
 {
   InetAddress adr = conn.getInetAddress();
   if (adr!=null) {
        unpairedWConns.remove(adr);
        return true;
   }
   return false;
 }        //removeStream

//METHODS CONCERNING DISCUSSIONS
 public Discussion addDiscussion (String user, String topic, String descript,
                    String parentstr, int roomcap,StreamManager output)
//adds new Discussion to Database
//parent = topic of parent discussion
//returns newly created discussion if it is not at MaxDiscs
//returns null if a new discussion can not be created
 {
  dprint("adding Discussion "+topic);
  if (NumDiscs()<MaxDiscs) {
    Discussion parent=findDiscussion(parentstr);
    if (parent==null) {
       System.err.println ("Error: parent discussion not found");
       return null;
     }

    //checking whether discussion name is already used
    Discussion NewGuy = findDiscussion (topic);
/*taken care of on client side now
    if (NewGuy!=null) {
      dprint ("discussion name already used");
      output.send(sc.USEDNAMESTRING,descript);
      return null;
    }
*/

    //everything's ok; adding discusssion
    NewGuy = new Discussion(topic,descript,ps,parent,roomcap);
    ps.broadcast (sc.ADDSUBDISC,parentstr,topic);
    DiscList.addElement(NewGuy);
    parent.addChild(user,NewGuy);
          findPerson(user).recordChild(user,NewGuy);
    return NewGuy;
   }      //if NumDiscs

   //NumDiscs >= MaxDiscs
   output.send(sc.MAXDISCSTRING);
   return null;
  }      //addDiscussion

  public void removeDiscussion (String discname)
  //kills the discussion
  //if there is someone inside it, throws DiscFullException
  //currently, not in use
  {
   dprint ("Removing discussion "+discname);
   Discussion d = findDiscussion(discname);
   if (d.getPeople().hasMoreElements())
    /*send the appropriate string to the user*/;
   else {
    if (d.getParent()!=null) d.getParent().removeChild(d);
//    d.saveHistory();
    DiscList.removeElement(d);
   }    //if d.People.size
  }  //removeDiscussion

  public boolean atCapacity() {return (NumPeople()>=MaxPeople);}

  public boolean isActive(String name)
  //returns true if there is a person who is in the conference under name
  {return findPerson(name)!=null;}

  public void setWorldSize (String user, String negDims, String posDims)
  //records outer boundaries of 3-D world for user's discussion
  //negDims = smallest/most negative x,y, & z coordinates occupied by any
  //        object in the room
  {
```

```
  Discussion d = findPerson(user).getDisc();
  dprint ("setting world size for discussion: "+d);
  d.setDims(negDims,posDims);
 }

//METHODS CONCERNING PEOPLE
 public void createPerson (String name, String pwd, PromptConnection bout)
  throws IOException
 //creates new Person in discussion room d
 //bout = prompt connection to client side for this person
 //value returned is this new Person
 {
  //pairing PromptConnections & WorldConnections
  //removing them from "unpaired" hashtables
  InetAddress adr = bout.getInetAddress();
  WorldConnection rout = (WorldConnection) unpairedWConns.get (adr);
  if (rout == null) {
   System.err.println
    ("Error: Paired request connection not found for "+adr);
   removeStreams (bout);
  }

  unpairedPConns.remove(bout);
  unpairedWConns.remove(adr);

  dprint ("connections paired");
  Person p = new Person (name, bout, rout);
  if (!Passwords.containsKey(name))
   //then this is the first time this user has logged in
   //send message to all users about new user
   ps.broadcast(sc.NEWUSERSTRING,name);
  String disc = p.loadHistory();
  dprint ("Adding person to database:"+p.toString());

  //adding appropriate ChatWindows to new user's desktop
  TalkSlot nextslot;
  for (Enumeration enum1=SlotList.elements(); enum1.hasMoreElements();) {
   nextslot = (TalkSlot) enum1.nextElement();
   p.p_out.send(sc.NEWWINSTRING,sc.CHATSTRING,nextslot.getSpeaker(),
        nextslot.getText());
  }      //for enum

  //adding person to database
  Passwords.put (name,pwd);
  if (disc==null) disc = sc.DEFAULTDISC;
  p.p_out.send (sc.CLEARDESKTOP,disc);
  findDiscussion(disc==null?sc.DEFAULTDISC:disc).addPerson (p);
  PersonList.addElement(p);
 }  //createPerson
```

```
public String loadPassword (String name)
//reads the password from some confidential file
//returns true if the password was found
{
 dprint ("loading password for "+name);
 return (String)Passwords.get(name);
}   //loadPassword

public void movePerson (String pers, String disc)
//moves p from present discussion to d
{
 dprint ("Moving "+pers+ " to "+disc);

 Person p = findPerson(pers);
 Discussion d = findDiscussion (disc);

 if (d.hasRoom()) {
  p.send (sc.CLEARDESKTOP,disc);
  p.getDisc().removePerson(p);
  d.addPerson(p);
 } //if d.hasRoom
 else p.p_out.send(sc.DISCFULLSTRING);
} //movePerson

public boolean destroyPerson (String input)
//used when person exits conference
//returns true if person was successfully removed
{
 Person p = findPerson (input);

 if (p==null) return false;
 dprint ("Destroying person "+p);
 p.w_out.send(sc.CLEARDESKTOP," ");
 p.writeLastDisc();
 p.getDisc().removePerson(p);
 removeSlot(p);
 savePwd(p);

 PersonList.removeElement(p);
 p.destroyConnections();
 return true;
}

public TalkSlot addSlot (String speaker)
//adds slot for speaker & creates new ChatWindow for all other users
//returns newly created slot
{
 Person p=findPerson(speaker);
 TalkSlot newslot = new TalkSlot(speaker);
```

```
    p.setCurSlot(newslot);
    SlotList.addElement(newslot);

    //adding new TalkWindow to all other people's desktops
    Person nextperson;
    for (Enumeration list = PersonList.elements(); list.hasMoreElements();)
      if (!(nextperson=(Person)list.nextElement()).equals(p))
        nextperson.p_out.send (sc.NEWWINSTRING,sc.CHATSTRING,
                        p.toString(),sc.EMPTYFIELD);
    return newslot;
  }      //addSlot

  public void removeSlot (String speaker)
  //speaker's SendChatWindow has been closed
  {
    Person p = findPerson(speaker);
    if (p==null)
      System.err.println
        ("Error: trying to remove slot of nonexistent person");
    else removeSlot(p);
  }

  public void removeSlot (Person p)
  {
    TalkSlot slot = p.getSlot();
    if (slot!=null) {
      SlotList.removeElement (slot);
      dprint ("traversing thru PersonList to remove ChatWindow: "+p);
      Person nextpers;
      for (Enumeration enum = PersonList.elements();
           enum.hasMoreElements();) {
             nextpers = (Person)enum.nextElement();
             if (!p.equals(nextpers))
         nextpers.p_out.send (sc.CLOSECHAT,p.toString());
      } //for enum
    } // if slot
//   p.getDisc().removeSlot(p);
  p.setCurSlot((TalkSlot)null);
  }    //removeSlot

//METHODS CONCERNING IMPORTS
  public void sendComment (String spkr, String text)
  //broadcasts what spkr sent thru a SendCommWindow
  {
    dprint ("Database:sendComm");
    Person p = findPerson(spkr);
    Discussion d = p.getDisc();
    String id=d.sendComment(p, text); //broadcasts comment to everyone
          //id = speaker + the id# for this comment
    Record r = new Record(sc.COMMENTSTRING,id,d.toString(),text);
```

```
 p.addRecord(r);
 d.addRecord(r);
 p.w_out.send(sc.ASKFORSIZE);
}  //sendComment

public void addImage (String user, String disc, String title, String address)
//adds image to given discussion
//user imported this image in disc
{
 Discussion d = findDiscussion(disc);
 Person pers = findPerson(user);
 d.addImage(title,address,pers);
 d.recordImage(user,disc,title,address);
 pers.recordImage(user,disc,title,address);
 pers.w_out.send(sc.ASKFORSIZE);
}   //addImage

public void add3DObj (String user, String disc, String title, String address)
//adds image to given discussion
{
 Discussion d = findDiscussion(disc);
 Person pers = findPerson(user);
 d.add3DObj(title,address,pers);
 d.record3DObj(user,disc,title,address);
 pers.record3DObj(user,disc,title,address);
 pers.w_out.send(sc.ASKFORSIZE);
}   //add3DObj

public void removeImport (String remover, String type, String id)
//removes import of given type in given discussion with given id
{
 Person pers = findPerson(remover);
 pers.getDisc().removeImport (pers,type,id);
}

public void moveObject (String pers,
                        String wintype, String win_id, String new_pos)
//notifies all user in discussion disc that the given window has been moved
//by person pers to position new_pos
{
 Person p = findPerson(pers);
 p.getDisc().moveObject(pers,wintype,win_id,new_pos);
 p.w_out.send(sc.ASKFORSIZE);
}  //moveObject

public void recordPos (String importtype, String user, String id, String pos)
//records position of given object
//importtype = comment, picture, etc.
//user = user who sent message, so the discussion in which to look for the
//        object can be determined
```

```
  //id = identifying string for import
  //pos = unparsed position
  {findPerson(user).getDisc().recordPos (importtype,id,pos);}

//METHODS CONCERNING WINDOWS IN PROMPTFRAME
  public void addHistWin (String opener, String disc, String histperson)
  //adds window for history of person in discussion disc to opener's Desktop
  {
   findPerson(opener).p_out.send (sc.NEWWINSTRING,sc.HISTWINSTRING,disc,
      histperson, getHistory(disc,histperson));
  } //addHistWin

  public void setChat (String spkr, String text)
  //broadcasts what spkr sent thru a SendChatWindow
  {
   dprint ("Database:setChat");

   Person p = findPerson(spkr);
   Discussion d = p.getDisc();
   Record r = new Record(sc.CHATSTRING,p,d,text);

   ps.broadcast (sc.SETTEXT,spkr,d.toString(),text);
//   ps.broadcast (sc.SETTEXT,spkr,text);
   p.getSlot().setText(text);
//   d.sendChat(spkr, text);
   p.addRecord(r);
   d.addRecord(r);
  }  //sendChat

//PRIVATE FUNCTIONS
  private void loadPasswords()
  //loads all passwords from PWDFILE
  //only invoked once -- when Database is constructed
  {
   dprint ("Loading passwords");
   RandomAccessFile pwdfile;
   try {
     pwdfile = new RandomAccessFile(PWDFILE,"r");
   } catch (IOException e) {
    System.out.println ("Problem opening password file for loading.");
    return;
   } //try/catch

   try {
    String name;
    String pwd;
    for (;pwdfile.getFilePointer() < pwdfile.length();) {
     Passwords.put (name=pwdfile.readLine(),pwd=pwdfile.readLine());
     dprint ("Password loaded: "+name+'.'+pwd);
    } //for pwdfile
    dprint ("Passwords:" + Passwords);
    pwdfile.close();
  } catch (IOException e) {
    System.err.println ("Problem reading password file while loading.");
    try {pwdfile.close();} catch (IOException e2) {}
  } //try/catch
} //loadPasswords()

private Person findPerson (String searchstr) {
  Person p;
  for (Enumeration list = PersonList.elements(); list.hasMoreElements();)
    if ((p=(Person) list.nextElement()).toString().equals(searchstr))
        return p;

  //for loop finished; topic not found
  dprint ("Person "+searchstr+" not found in database.");
  return null;
}  //findPerson

private Discussion findDiscussion (String searchstr) {
  Discussion d;
  for (Enumeration list = DiscList.elements(); list.hasMoreElements();)
    if ((d=(Discussion) list.nextElement()).toString().equals(searchstr))
        return d;

  //for loop finished; searchstr not found
  System.out.println ("Discussion "+searchstr+" not found in database.");
  return null;
}       //findDiscussion

private String getHistory (String d, String p)
//loads history for appropriate person & discussion
//doesn't actually load from disk -- uses history stored in memory

/*The strategy now is to load from disk when the Recorder is created, then
send it when the user requests it.  Although this technique allows for a
fast response to a request for a history window, it eats up a huge chunk
of memory.  Since users will not be expected to read from disk that much,
in the future, the History should not be kept in memory.  Instead, it should
be read from disk in this procedure, then sent to the user.
*/

{
  Person pers=null;
  Discussion disc=null;
  StringBuffer histbuffer = new StringBuffer();
  Record nextrec;
  Vector history;

  if (!p.equals(sc.EMPTYFIELD))
  if ((pers=findPerson (p))==null)  //if person not currently in conference
    pers = new Person (p);  //create temporary person
 if (!d.equals(sc.EMPTYFIELD))
  if ((disc = findDiscussion (d))==null)
    disc = new Discussion (d);

 dprint ("LOADING HISTORY . . .");
 if (pers!=null)
  for (Enumeration histenum=pers.getHistory();
        histenum.hasMoreElements();)
  {
    nextrec=(Record)histenum.nextElement();
    if (disc==null)
      histbuffer.append(nextrec.Topic+": "+nextrec.Quote+"\n"+
                  nextrec.TimeStamp+"\n\n");
    else if (disc.getTopic().equals(nextrec.getTopic()))
      histbuffer.append (nextrec.Quote+"\n"+nextrec.TimeStamp+"\n\n");
  }         //for Enumeration histenum
 else  //if pers
  for (Enumeration histenum=disc.getHistory();
        histenum.hasMoreElements();)
  {
    nextrec=(Record) histenum.nextElement();
    histbuffer.append (nextrec.Speaker+": "+nextrec.Quote+"\n"+
                  nextrec.TimeStamp+"\n\n");
  }   //for Enumeration histenum
 String returnstr = histbuffer.toString();
 dprint (returnstr);
 return returnstr.equals("")?sc.EMPTYFIELD:returnstr;
}           //getHistory

private boolean savePwd(Person p)
//saves password
{
 dprint
   ("saving password: "+p+","+Passwords.get(p.toString()));
 Passwords.put (p.toString(),Passwords.get(p.toString()));

 try {
  RandomAccessFile pwdfile = new RandomAccessFile(PWDFILE,"rw");
  try {
    String name = p.getName();
    for (;pwdfile.getFilePointer() < pwdfile.length();)
      if (pwdfile.readLine().equals(name)) {
        /*write password on next line -- currently, there is no way
        to change one's password*/
             return true;
      } //if pwdfile
      else pwdfile.readLine();   //skip over password
    dprint ("Name not found; creating new entry.");
```

```
        pwdfile.writeBytes(name+'\n');
        pwdfile.writeBytes((String)Passwords.get(name)+'\n');
        pwdfile.close();
        return true;
      } catch (IOException e) {
        System.err.println ("Problem writing to password file while saving.");
        pwdfile.close();
        return false;
      } //try/catch
    } catch (IOException e) {
      System.out.println ("Problem opening password file for saving.");
    } //try/catch
    return false;
  }     //savePwd

  private void dprint (String msg) {
    //prints to primary output if DEBUG flag is on
    if (DEBUG) System.out.println (msg);
  }

} //Database
```

---

```
/* Discussion.java
contains Discussion class.

Discussion is a subclass of Recorder
It contains all the information about a particular discussion that has
been created somewhere in DDS.
*/

//package DatabasePack;
import java.awt.*;
import java.io.*;
import java.util.*;
import ConferenceServer;
import StringCode;
import PromptServer;
//import StreamManager;
import ImportInfo;

class Discussion extends Recorder {
//CONSTANTS
  private final static boolean DEBUG=Database.DEBUG;

//FIELDS
  protected int MaxPeople = 10;
  int NumPeople = 0;
  Discussion Parent;
  Vector Children = new Vector (2,2);  //vector of discussions
  String Topic;  //name of topic
  String Description;  //brief description of topic
  String negDims = sc.DEFAULTDISCDIMS;
            //dimensions of discussion (read from file)
            //held in a string form that can be parsed by client
            //negDims indicates the smallest/most negative coordinates at which
            //an object exists in this discussion's representation of the world
  String posDims = sc.DEFAULTDISCDIMS;

  Vector People = new Vector (2,1);  //people in the discussion
//  Vector SlotList = new Vector (2,1);
  Hashtable Images = new Hashtable(2,(float)1);  //images in discussion
  Hashtable vrmlObjs = new Hashtable(2,(float)1);  //3D vrmlObjs in discussion
//  Vector Layout = new Vector (5,2); //list of window defaults
  protected Hashtable Comments = new Hashtable (3);          //list of CommentWindows
  static PromptServer ps;

  public Discussion (String topic)
  //used to create temporary discussions when loading history
  {
    super (topic);
    dprint ("creating temporary discussion "+topic);
  }

  public Discussion (String topic, String descript, PromptServer b,
                              Discussion parent, int max)
//  throws UsedNameException
  //if checkUsedName is true, checks whether discussion topic given has
            //already been used before
  {
    super (topic);    //sets ShortName
    dprint("creating new Discussion "+topic);
    Topic = topic;
    Description = descript;
    ps = b;
    Parent = parent;
    MaxPeople = max;

    loadHistory(Images,vrmlObjs,Comments,Children,ps);
                        //this method will temporarily put Strings for the values
                        //of Images, vrmlObjs, and Comments
                        //in loadLayout(), the Strings are replaced with ImportInfos
    Vector layout = loadLayout();
    incorporateLayout (layout);

  }  //Discussion constructor

//PUBLIC FUNCTIONS
  public Discussion getParent () {return Parent;}
  public String getTopic() {return Topic;}
  public String getDescription() {return Description;}
```

```
 public String getNegDims () {return negDims;}
 public String getPosDims () {return posDims;}
 public Enumeration getPeople() {return People.elements();}
// public Enumeration  getImages() {return Images.elements();}
// public Enumeration getSlotList() {return SlotList.elements();}
 public Enumeration getHistory() {return super.getHistory();}
 public Vector getChildren() {
//   if (Children.size()>0)
   return Children;
//   else return null;
 }
 public String toString() {return Topic;}

 public boolean hasRoom() {return NumPeople<MaxPeople;}

 public Vector getDescendants ()
 //recursive procedure that returns a Vector of all children, grandchildren,
 //etc. of this discussion
 {
  Vector returnvec = (Vector)Children.clone();

  //ask each child for its descendants
  for (Enumeration enum = Children.elements(); enum.hasMoreElements();) {
   Vector descendants = ((Discussion)enum.nextElement()).getDescendants();

   //add all discussions in descendants to returnvec
   for (Enumeration enum2 = descendants.elements();
        enum2.hasMoreElements();)
    returnvec.addElement(enum2.nextElement());
  }   //for Enumeration enum

  dprint ("finding descendants of "+Topic+':'+returnvec);
  return returnvec;
 }   //getDescendants

 public boolean setDims (String newneg, String newpos)
 //sets dims to given value, and updates layout file
 //returns true if dims is successfully changed
 {
  dprint ("setting dims for discussion:"+Topic);
  negDims = newneg;
  posDims = newpos;
  return saveLayout();
 }  //setDims

 public void addChild (String user, Discussion child)
 //adds child to Children and sends message to all people in discussion to
 // add a subdiscussion button to their desktop
 //also writes to history file that a subdiscussion was created
 {
   Children.addElement(child);
   notifyWorlds (sc.ADDSUBDISC,child.getTopic(),sc.DEFAULTDISCDIMS,
                    sc.DEFAULTDISCDIMS);
   recordChild (user, child);
  }      //addChild

  public void removeChild (Discussion child) {
   Children.removeElement(child);
//   for (Enumeration list = People.elements(); list.hasMoreElements();)
//     ((Person)list.nextElement()).removeDisc(child.getTopic());
  } //removeChild

  public void addImage (String title, String adr, Person importer) {
   System.out.println ("Adding image to discussion "+Topic+": "+adr);
   Images.put(title,new ImportInfo(adr,title));

   //notify promptframes of users in this discussion
   //a list of all imports is kept in the promptframe so users can choose
   //which object to remove, if they want to remove one
   notifyPromptframes (sc.PICWINSTRING, title, adr);

   //update worlds of users in this discussion, but don't ask them
   //reply by using sc.EMPTYFIELD instead of sc.REPLY
   notifyWorldsExcept (importer.getName(),sc.PICWINSTRING,title,
                              adr,sc.EMPTYFIELD);

   //update world of user who imported picture, and ask him to reply
   importer.w_out.send(sc.PICWINSTRING,title,adr,sc.REPLY);
  }    //addImage

  public void add3DObj (String title, String adr, Person importer) {
   System.out.println ("Adding object to discussion "+Topic+": "+adr);
   vrmlObjs.put(title,new ImportInfo(adr,title));

   //notify promptframes of users in this discussion
   //a list of all imports is kept in the promptframe so users can choose
   //which object to remove, if they want to remove one
   notifyPromptframes (sc.OBJSTRING, title, adr);

   //update worlds of users in this discussion, but don't ask them
   //reply by using sc.EMPTYFIELD instead of sc.REPLY
   notifyWorldsExcept (importer.getName(),sc.OBJSTRING,title,
                              adr,sc.EMPTYFIELD);

   //update world of user who imported picture, and ask him to reply
   importer.w_out.send(sc.OBJSTRING,title,adr,sc.REPLY);
  }    //add3DObj

  public void removeImport (Person remover, String type, String id) {
   System.out.println ("Removing import from "+Topic+':'+id);
  //update memory and disk
  findImportList(type).remove(id);
  saveLayout();

  //tell other users
  if (type.equals(sc.COMMENTSTRING)) {
  //broadcast to all users so they can update history windows, if necessary
    ps.broadcast (sc.REMOVECOMM,Topic,id);

    //remove comment in worlds of users in discussion
    notifyWorlds (sc.REMOVEIMPORT,sc.COMMENTSTRING,id);
  }
  else
    //send message to all promptframes and worlds of users in this
    //discussion
    notifyAll (sc.REMOVEIMPORT, type, id);

  //get new size of discussion
  remover.w_out.send (sc.ASKFORSIZE);

  //record removal to disk
  Discussion d = remover.getDisc();
  Record r = new Record(sc.REMOVEIMPORT,remover,d,id);
  remover.addRecord(r);
  d.addRecord(r);
}   //removeImage

public void addPerson (Person dude)
//note that a person does NOT get a talkslot unless if he requests one
{
  dprint ("adding "+dude+" to "+toString());
  if (NumPeople<MaxPeople) {
    People.addElement(dude);
    dude.setCurDisc(this);

    String str1, str2, defs;
    ImportInfo info;          //info of next Import to be examined
    //loading CommentWindows
    Hashtable commCopy = (Hashtable)Comments.clone();
                              //running list of comments
                              //made by dude.  As we come across comments that
                              //aren't made by him, this Hashtable gets smaller
    Enumeration enum1=Comments.keys();
    Enumeration enum2=Comments.elements();
    for (;enum1.hasMoreElements();) {
      str1=(String)enum1.nextElement();  //speaker & id #
      info=(ImportInfo)enum2.nextElement();
          str2 = info.getValue();                    //comment
          defs = info.getPos();
```

```
        //at the

        dude.w_out.send(sc.COMMENTSTRING,str1,str2,defs);

        //simultaneously, figure out which comments were made by this speaker
        //then send him the list of his comments
        if (!StreamManager.parseCommIdGetSpkr(str1).equals(dude.getName()))
          //this comment wasn't made by him; remove it from commCopy
          commCopy.remove(str1);
}       //for enum
//now returned shortened version of Comments
dude.p_out.send(sc.COMMENTLIST,
                StreamManager.unparseList(commCopy.keys()));

//loading Images
enum1 = Images.keys();
for (enum2 = Images.elements(); enum1.hasMoreElements();) {
  str1 = (String) enum1.nextElement();  //title
  info = (ImportInfo) enum2.nextElement();
        str2 = info.getValue();                 //url
        defs = info.getPos();

  dude.w_out.send(sc.PICWINSTRING,str1,str2,defs);
        dude.p_out.send(sc.PICWINSTRING,str1,str2);
}       //for enum

//loading 3-D vrmlObjs
enum1 = vrmlObjs.keys();
for (enum2 = vrmlObjs.elements(); enum1.hasMoreElements();) {
  str1 = (String) enum1.nextElement();  //title
  info = (ImportInfo) enum2.nextElement();
        str2 = info.getValue();                 //url
        defs = info.getPos();

  dude.w_out.send(sc.OBJSTRING,str1,str2,defs);
  dude.p_out.send(sc.OBJSTRING,str1,str2);
}       //for enum

//loading subdiscussions & superdiscussion
if (Parent!=null) dude.send(sc.ADDSUPERDISC,Parent.toString());
for (enum1=Children.elements(); enum1.hasMoreElements();) {
        Discussion d = (Discussion)enum1.nextElement();
  str1 = d.getTopic();              //topic of subdiscussion
        str2 = d.getNegDims();                  //dimensions of subdiscussion,
        String str3 = d.getPosDims();  //with sc.IDDIVIDER between width,
                              // height, & depth (parsed on client side)
  dude.w_out.send(sc.ADDSUBDISC,str1,str2,str3);
        dude.p_out.send(sc.ADDSUBDISC,Topic,str1);
        //sending imports of this subdiscussion
        //first, the comments
        Enumeration enum = d.Comments.elements();
        while (enum.hasMoreElements()) {
          info = (ImportInfo)enum.nextElement();
          dude.w_out.send (sc.ADDPROTO,sc.COMMENTSTRING,d.Topic,
                         sc.EMPTYFIELD,                //this field is the url, but
                                                     //comments don't have urls
                         info.getPos());
        }

        //next, the Images
        for (enum=d.Images.elements();enum.hasMoreElements();) {
          info = (ImportInfo)enum.nextElement();
          dude.w_out.send (sc.ADDPROTO,sc.PICWINSTRING,d.Topic,
                         info.getValue(),  //url
                         info.getPos());
        }        //for enum

        //next, the vrmlObjs
        for (enum=d.vrmlObjs.elements();enum.hasMoreElements();) {
          info = (ImportInfo)enum.nextElement();
          dude.w_out.send (sc.ADDPROTO,sc.OBJSTRING,d.Topic,
                         info.getValue(),  //url
                         info.getPos());
        }        //for enum
      }      //for slotenum

/*      NOW TAKEN CARE OF BY DATABASE
      //loading ChatWindows
      TalkSlot nextslot;
      for (enum1=SlotList.elements(); enum1.hasMoreElements();) {
        nextslot = (TalkSlot) enum1.nextElement();
        dude.send(sc.NEWWINSTRING,sc.CHATSTRING,nextslot.getSpeaker(),
                   nextslot.getText());
      }      //for enum
*/
/*      NO LONGER CHECKED
      if (notAdded.size() > 0)
        System.err.println ("Error: not all elements in WinLayout have been added");
*/
    } //if NumPeople
    else
      System.err.println ("Error: trying to add person to discussion that is"
                   +"already at capacity");
  }      //addPerson

  public void removePerson (Person dude) {
    dprint ("removing "+dude+" from "+Topic);
    People.removeElement(dude);
    dude.setCurDisc(null);
//    removeSlot (dude);
  } //removePerson

  public String sendComment (Person pers, String text)
  //text was sent by pers thru SendCommWindow
  //this procedure is used for brand new comments
  //returns identifier for created comment.  This means the speaker,
  //plus the id # for the comment.
  {
    String spkr = pers.getName();
    dprint
           ("traversing thru People to create comment window: "+spkr);

    int idnum = pers.incCommId();  //to distinguish this comment from
                     //other comments by the same speaker
    String identifiers = StreamManager.unparseCommId(spkr,idnum);
    Comments.put (identifiers, new ImportInfo (text,identifiers));

    //notify all promptframes (even those for people outside the discussion)
    //so that users can update their history windows, if they have any open
    ps.broadcast (sc.COMMENTSTRING,toString(),identifiers,text);

    //notify all worlds of users in this discussion, but don't tell them
    //to reply by using sc.EMPTYFIELD (and not using sc.REPLY)
    //this is done to reduce network traffic
    notifyWorldsExcept (spkr,sc.COMMENTSTRING,identifiers,text,sc.EMPTYFIELD);

    //person who posted comment should reply to tell server where comment is
    //placed
    pers.w_out.send (sc.COMMENTSTRING,identifiers,text,sc.REPLY);

    return identifiers;
  }        //sendComment

  public void moveObject (String mover, String wintype, String win_id,
                          String new_pos)
  //notify all users in this discussion of object moved by mover
  //object can be an import or a comment
  //updates layout, both in memory and on disk
  {
    //tell users (except for the one who moved it)
    if (DEBUG)
      System.out.println ("Notifying users in "+Topic+" of moved object");
    notifyWorldsExcept(mover,sc.OBJMOVEDSTRING,wintype,win_id,new_pos);

    //update memory
    if (wintype.equals(sc.COMMENTSTRING))
      ((ImportInfo)Comments.get (win_id)).setPos(new_pos);
    else if (wintype.equals(sc.PICWINSTRING))
    ((ImportInfo)Images.get (win_id)).setPos(new_pos);
   if (wintype.equals(sc.OBJSTRING))
    ((ImportInfo)vrmlObjs.get (win_id)).setPos(new_pos);

   //update disk
   saveLayout();
  } //moveObject

  public void recordPos (String importtype, String id, String pos)
  //records position of given import, identified by id, of type importtype
  //        both in memory and on disk
  {
   dprint ("recording position of "+id);
   if (importtype.equals(sc.COMMENTSTRING))
    ((ImportInfo)Comments.get(id)).setPos(pos);
   else if (importtype.equals(sc.PICWINSTRING))
    ((ImportInfo)Images.get(id)).setPos(pos);
   else if (importtype.equals(sc.OBJSTRING))
    ((ImportInfo)vrmlObjs.get(id)).setPos(pos);

   //save to disk
   saveLayout();
  }        //recordPos


//PRIVATE METHODS
  private void notifyAll (String s1, String s2, String s3)
  {notifyAll (StreamManager.unparseMessage (s1,s2,s3));}

  private void notifyAll (String s1, String s2)
  {notifyAll (StreamManager.unparseMessage (s1,s2));}

  private void notifyAll (String msg)
  //sends mesage to all users in discussion, both through WorldConnection
  //   and through PromptConnection
  {
   dprint ("sending to all users in "+Topic+':'+msg);
   Person nextpers;
   for (Enumeration list = People.elements(); list.hasMoreElements();) {
    nextpers = (Person)list.nextElement();
    nextpers.send(msg);
   }
  } //notifyAll

  private void notifyWorlds (String s1, String s2)
  {notifyWorlds (StreamManager.unparseMessage(s1,s2));}

  private void notifyWorlds (String s1, String s2, String s3)
  {notifyWorlds (StreamManager.unparseMessage(s1,s2,s3));}
```

```
private void notifyWorlds (String s1, String s2, String s3, String s4)
{
 notifyWorlds (StreamManager.unparseMessage(s1,s2,s3,s4));
}

private void notifyWorlds (String msg)
//sends msg through WorldConnections of people in this discussion
{
 dprint ("sending to all worlds in "+Topic+':'+msg);
 Person nextpers;
 for (Enumeration list = People.elements(); list.hasMoreElements();) {
  nextpers = (Person)list.nextElement();
  nextpers.w_out.send(msg);
 }
}  //notifyWorlds

private void notifyWorldsExcept (String outcast, String s1, String s2,
                    String s3, String s4)
{notifyWorldsExcept (outcast, StreamManager.unparseMessage (s1,s2,s3,s4));}

private void notifyWorldsExcept (String outcast, String msg)
//sends msg through WorldConnections of people in this discussion
//except for the person with the name of outcast
{
 dprint ("sending to all worlds in "+Topic+" except "+outcast+':'+msg);
 Person nextpers;
 for (Enumeration list = People.elements(); list.hasMoreElements();) {
  nextpers = (Person)list.nextElement();
  if (!nextpers.getName().equals(outcast))
   nextpers.w_out.send(msg);
 }      //for list
}  //notifyWorldsExcept

private void notifyPromptframes (String s1, String s2, String s3)
{notifyPromptframes(StreamManager.unparseMessage(s1,s2,s3));}

private void notifyPromptframes (String msg)
//sends msg through PromptConnections of people in this discussion
{
 dprint ("sending to all promptframes in "+Topic+':'+msg);
 Person nextpers;
 for (Enumeration list = People.elements(); list.hasMoreElements();) {
  nextpers = (Person)list.nextElement();
  nextpers.p_out.send(msg);
 }
}  //notifyPromptframes

private Vector loadLayout ()
//loads layout & size for this discussion
//returns a vector of all the defaults in the file
```

```
{
 dprint ("loading window layout: "+Topic);
 Vector returnVec = new Vector (4);

 //open layout file
 RandomAccessFile f;
 try { f = new RandomAccessFile (Topic+".layout","r");
      dprint("Regular layout file opened");
    } catch (IOException e) {
      System.err.println ("Trouble opening layout file in "+Topic);
          return returnVec;
    }   //try f ... Topic ... / catch


 //read layout line by line, putting each line into the appropriate
 //Hashtable.  Note that saveLayout guarantees that all records in the
 //layout file start with negDims and posDims, then the Comments, then
 //the Images, then the 3D vrmlObjs
 try {
   //first, read in dimensions of discussion
   negDims = f.readLine();
   posDims = f.readLine();

   //then, read in defaults
   for (String nextline = f.readLine();
                  !nextline.equals(sc.DONESTRING);
                  nextline = f.readLine()) {
        if (nextline==null) {        //end of file has been reached,
                                     //and no DONESTRING found
          System.err.println ("ERROR: layout file in wrong format");
          return new Vector (1);
        }                    //if nextline
        returnVec.addElement(nextline);
    }              //for nextline

/*NOW TAKEN CARE OF (PARTIALLY) BY incorporateLayout
   String nextline = f.readLine();

   String nextId = ImportInfo.getId(nextline);

   //first, traverse through Comments
   try {
     while (!nextline.equals(sc.DONESTRING)) {
          //will break out of loop upon NullPointerException
          ((ImportInfo)Comments.get(nextId)).setDefaults(nextline);
          nextline = f.readLine();
          nextId = ImportInfo.getId(nextline);
        }                    //while nextline
    } catch (NullPointerException e) {
        //done with set defaults in Comments; move on to Images;
     }

     //then, through Images
     try {
       while (!nextline.equals(sc.DONESTRING)) {
            //will break out of loop upon NullPointerException
            ((ImportInfo)Images.get(nextId)).setDefaults(nextline);
            nextline = f.readLine();
            nextId = ImportInfo.getId(nextline);
          }
     } catch (NullPointerException e) {
          //done with set defaults in Comments; move on to Images;
     }

     //then, through vrmlObjs
     while (!nextline.equals(sc.DONESTRING)) {
          ((ImportInfo)vrmlObjs.get(nextId)).setDefaults(nextline);
          nextline = f.readLine();
          nextId = ImportInfo.getId(nextline);
     }
*/

     //should be done
     f.close();
    } catch (IOException e) {
     System.err.println ("Couldn't read "+Topic+"'s layout file.");
     try {
       f.close();
     } catch (IOException e2) {
          System.err.println ("trouble closing file.");
     }
    }  //try/catch
    dprint ("done loading layout");
    return returnVec;
  }  //loadLayout

  private void incorporateLayout (Vector layout)
  //modifies: this.Comments, this.Images, this.vrmlObjs
  //scans through layout, and modifies the ImportInfos of Comments,
  //Images, and vrmlObjs appropriately
  //Note that saveLayout guarantees that all records in the
  //layout file start with negDims and posDims, then the Comments, then
  //the Images, then the 3D vrmlObjs
  {
    dprint ("incorporating layout");

    Enumeration layoutEnum = layout.elements();
    String curObj;
    try {             //breaks out of this try upon a NoSuchElementException
                              //i.e., when the end of layout is reached
    String nextline = (String)layoutEnum.nextElement();
    String nextId = ImportInfo.getId(nextline);

    //first, traverse through Comments
    dprint ("traversing through comments");
    ImportInfo nextInfo = (ImportInfo)Comments.get(nextId);
    for (;nextInfo!=null;nextInfo=(ImportInfo)Comments.get(nextId)) {
        dprint ("nextline:"+nextline);
        nextInfo.setDefaults(nextline);              //change ImportInfo's default
        nextline = (String)layoutEnum.nextElement();     //go to next default
        nextId = ImportInfo.getId(nextline);         //find out which object
                                                 //we're talking about
    }            //while nextline

    //done with set defaults in Comments; move on to Images;
    dprint ("traversing through Images");
    for (nextInfo = (ImportInfo)Images.get(nextId);
            nextInfo!=null;
            nextInfo = (ImportInfo)Images.get(nextId)) {
        dprint ("nextline:"+nextline);
        nextInfo.setDefaults(nextline);              //change ImportInfo's default
        nextline = (String)layoutEnum.nextElement();     //go to next default
        nextId = ImportInfo.getId(nextline);         //find out which object
                                                 //we're talking about
    }            //while nextline

    //next, vrmlObjs
    dprint ("traversing through vrmlObjs");
    for (nextInfo = (ImportInfo)vrmlObjs.get(nextId);
            nextInfo!=null;
            nextInfo = (ImportInfo)vrmlObjs.get(nextId)) {
        dprint ("nextline:"+nextline);
        nextInfo.setDefaults(nextline);              //change ImportInfo's default
        nextline = (String)layoutEnum.nextElement();     //go to next default
        nextId = ImportInfo.getId(nextline);         //find out which object
                                                 //we're talking about

    }            //while nextline
   } catch (NoSuchElementException e) { } //end of layoutEnum was reached
//    dprint ("nextline at the end:"+nextline);

/*NO LONGER NEEDED
    //any Comments left without defaults have been removed from the world
    //because they are removed from the layout file when a user removes them
    Enumeration listKeys = Comments.keys();
    Enumeration listInfos = Comments.elements();
    for (;listKeys.hasMoreElements();)
        if (((ImportInfo)listInfos.nextElement()).
                getDefaults().equals(sc.EMPTYFIELD))
        //then there's no default for this import
```

```
          Comments.remove(listKeys.nextElement());
        else
        //move on to next ImportInfo
          listKeys.nextElement();

    //any Images left without defaults have been removed from the world
    //because they are removed from the layout file when a user removes them
    listKeys = Images.keys();
    for (listInfos = Images.elements();listKeys.hasMoreElements();)
        if (((ImportInfo)listInfos.nextElement()).
                getDefaults().equals(sc.EMPTYFIELD))
        //then there's no default for this import
          Images.remove(listKeys.nextElement());
        else
        //move on to next ImportInfo
          listKeys.nextElement();

    //any 3D objs left without defaults have been removed from the world
    //because they are removed from the layout file when a user removes them
    listKeys = vrmlObjs.keys();
    for (listInfos = vrmlObjs.elements();listKeys.hasMoreElements();)
        if (((ImportInfo)listInfos.nextElement()).
                getDefaults().equals(sc.EMPTYFIELD))
        //then there's no default for this import
          vrmlObjs.remove(listKeys.nextElement());
        else
        //move on to next ImportInfo
          listKeys.nextElement();
*/

   if (layoutEnum.hasMoreElements()) {
        //not all layouts were placed
    System.err.println ("ERROR: Not all elements of layout were placed");
   }                           //if layoutEnum
  }                    //incorporateLayout


  private boolean saveLayout()
  //traverses through Comments, Images, and vrmlObjs, and saves
  //defaults to disk
  //format is as follows:
  //  1st line: negDims, unparsed appropriately
  //  2nd line: posdims
  //  next x lines (x>=0): defaults for Comments
  //  next y lines (y>=0): defaults for Images
  //  next z lines (z>=0): defaults for vrmlObjs
  {
   dprint ("saving layout for "+Topic);

   //open layout file
  RandomAccessFile f;
  try { f = new RandomAccessFile (Topic+".layout","rw");
      System.out.println("Regular layout file opened");
     } catch (IOException e) {
      System.out.println ("Trouble opening layout file in"+Topic);
          return false;
  }  //try f ... Topic ... / catch

  try {
   //write dims
   f.writeBytes (negDims+'\n'+posDims+'\n');

   //write rest of layout
   String nextdef;          //next default
   Enumeration enum;    //used to traverse through Hashtable elements

   //traverse through the Hashtables
   for (enum = Comments.elements(); enum.hasMoreElements();) {
         nextdef = ((ImportInfo) enum.nextElement()).getDefaults();
    f.writeBytes(nextdef+'\n');
   }     //for enum
   for (enum = Images.elements(); enum.hasMoreElements();) {
         nextdef = ((ImportInfo) enum.nextElement()).getDefaults();
    f.writeBytes(nextdef+'\n');
   }     //for enum
   for (enum = vrmlObjs.elements(); enum.hasMoreElements();) {
         nextdef = ((ImportInfo) enum.nextElement()).getDefaults();
    f.writeBytes(nextdef+'\n');
   }     //for enum

   //write DONESTRING just in case new dimensions
   //take up less space than old dimensions (in that case,
   //there would be residue at the end of the file)
   f.writeBytes (sc.DONESTRING+'\n');
   f.close();
   return true;
  } catch (IOException e) {
   System.err.println ("Trouble writing to layout file in"+Topic);
   return false;
  }                    //try/catch
}

private Hashtable findImportList (String list)
//returns Comments, Images, or vrmlObjs
{
 if (list.equals(sc.COMMENTSTRING)) return Comments;
 if (list.equals(sc.PICWINSTRING)) return Images;
 if (list.equals(sc.OBJSTRING)) return vrmlObjs;
 else System.err.println ("Error: Import list type not found:"
      +list);
 return null;
}                       //findImportList
} //Discussion
```

---

```
/* ImportInfo.java
contains ImportInfo class

This class is basically a data structure that contains information about
imports.  Currently, it holds the url (for pictures & 3D objects) or the
text (for comments), and the import's default position.  The Images, Objects,
and CommentList Hashtables of each Discussion should map titles of the imports
to ImportInfos.  Only the defaults field may be modified after an ImportInfo is
constructed.
*/

import StringCode;
import java.util.StringTokenizer;

public class ImportInfo {
//CONSTANTS
public static final boolean DEBUG = true;
public static final StringCode sc = new StringCode();
public static final int X = 0;
public static final int Y = 1;
public static final int Z = 2;

//FIELDS
String id = sc.EMPTYFIELD;
                        //identifier for corresponding object
String pos = sc.EMPTYFIELD;; //unparsed position of object
String value;       //url for pictures and 3-D objects;
                        //for comments, the comment text

//METHODS
public ImportInfo (String v, String i)
{
 value=v;
 id = i;
}

public ImportInfo (String v)
{value = v;}

public String getValue() {return value;}

public String getDefaults()
//returns id & pos, with PARAMDIVIDER in between
//if pos = EMPTYFIELD, returns EMPTYFIELD
{return pos.equals(sc.EMPTYFIELD)?id:id+sc.PARAMDIVIDER+pos;}
public void setDefaults(String d)
{
 StringTokenizer tok = new StringTokenizer(d,sc.PARAMDIVIDER,false);
 id = tok.nextToken();
 pos = tok.nextToken();
}

public String getId() {return id;}

public String getPos()
{return pos;}

public void setPos(String p) {pos =p;}

//STATIC METHODS
public static String getId(String s)
{
 StringTokenizer tok = new StringTokenizer(s,sc.PARAMDIVIDER,false);
 return tok.nextToken();
}
} // ObjInfo
```

---

```
/*Person.java
includes Person class.

Person class is the server-side representation of each user. Each time a
user logs in, a Person object is created to represent him. Person objects
are destroyed when users log out. The class contains information such
as the current discussion and information about communication with
the client machine with the user.
*/

import java.awt.*;
import java.util.*;
import java.io.*;
import StreamManager;

public class Person extends Recorder {
//CONSTANTS
  public final static boolean DEBUG =Database.DEBUG;

//FIELDS
  String Name;
  Discussion CurDisc; //discussion person is currently in
  TalkSlot CurSlot;//slot into which person is currently chatting
  protected StreamManager p_out; //prompt output for this person
  protected StreamManager w_out; //world output for this person
  PromptConnection pconn;
  WorldConnection wconn;
```

```
//CONSTRUCTORS
public Person (String n)
//temporary object used to load history
{
 super(n);
 loadHistory();
 if (DEBUG) System.out.println ("Creating temporary person "+n);
}

public Person (String n, PromptConnection p, WorldConnection w)
{
 super(n);
 if (DEBUG) System.out.println ("Creating person "+n);
 Name = n;
 pconn = p;
 wconn = w;
 p_out = p.output;
 w_out = w.output;
 wconn.getInputLoop().set_name(n);  //input loop of pconn should have already
                                    //been set
}

//PUBLIC FUNCTIONS
 public String getName() {return Name;}
 public Discussion getDisc() {return CurDisc;}
 public TalkSlot getSlot() {return CurSlot;}
 public Enumeration getHistory() {return super.getHistory();}

 public String toString () {return Name;}

 public void setCurDisc (Discussion d)
 //sets CurDisc to d
 //also adds it to PrevDiscs if it's not already there
 {CurDisc = d;}   //setCurDisc

 public void setCurSlot (TalkSlot s) {CurSlot = s;}

 public int incCommId ()
 //returns current value of commId and increments it
 {
  commId++;
  return commId-1;        //"-1" yields old value
 }                   //incCommId

 public boolean writeLastDisc()
 //writes at end the last discussion visited by this person
 {
   if (DEBUG) System.out.println ("Writing DONE for "+ShortName);
   try {
    RandomAccessFile outputfile = new RandomAccessFile (HistFile(),"rw");
    try {
          if (HistLine==WRITETOEND) outputfile.seek(outputfile.length());
          else for (int i=0;i<HistLine;++i) outputfile.readLine();
          outputfile.writeBytes(sc.DONESTRING+'\n');
          outputfile.writeBytes(CurDisc.toString()+'\n');
          outputfile.close();
    } catch (IOException e) {
      System.err.println ("Problem writing to history file: ");
          outputfile.close();
          return false;
    } //try/catch
   } catch (IOException e) {
    System.out.println ("Problem opening history file for writing.");
    return false;
   }  //try/catch
  return true;
 } //writeLastDisc

 public void destroyConnections() {
  pconn.destroy();
  wconn.destroy();
 }

 public void send (String message)
 //sends message through both prompt and world outputs
 {
  if (DEBUG) System.out.println ("sending for "+Name);
  w_out.send(message);
  p_out.send(message);
 }

 public void send (String m1, String m2)
 {send (StreamManager.unparseMessage(m1,m2));}

 public void send (String m1, String m2, String m3)
 {send (StreamManager.unparseMessage(m1,m2,m3));}

 public void send (String m1, String m2, String m3, String m4)
 {send (StreamManager.unparseMessage(m1,m2,m3));}
}   //Person
```

---

```
/* PromptServer.java
includes classes PromptServer, PSInLoop, and PromptConnection

The PromptServer is contacted when a client starts up DDS.  It opens a
PromptConnection with the client.  The PromptServer is used to communicate
with the PromptFrame of the client.  The PSInLoop waits for
messages from the client.
```

```
*/

import java.net.*;
import java.io.*;
import java.util.*;
import ServerConnection;
import InputLoop;
import StringCode;
import StreamManager;
import Database;

public class PromptServer extends Thread
{
//CONSTANTS
  private final static StringCode sc = new StringCode();
  protected final static boolean DEBUG = true;

// FIELDS
  private static int port;
  private static Vector connections = new Vector(2,1);
//  private Vector outputs = new Vector(2,1);
  private ServerSocket socket;
  public static Database database;

// METHODS
  public PromptServer(int _port)
  {port = _port;}

  public void run()
  {
    try {socket = new ServerSocket(port);}
    catch(Exception e) {
      StreamManager.showError(e);
      System.exit(1);
    }

    ServerConnection temp_sc;
    while (true)
    {
      try
      {
        System.out.println("PromptServer:Waiting for connection");
        Socket this_connection = socket.accept();
                System.out.println("PromptServer Connection!");

        // spin off new thread to handle new broadcast connection
        temp_sc = new PromptConnection(this_connection, this, database);
        temp_sc.start();
        connections.addElement(temp_sc);
//        outputs.addElement(temp_sc.output);
```

```
      //clean up the vector if needed
/*
      for(int i=0;i<PromptServer.connections.size();i++)
        if(!((ServerConnection)(connections.elementAt(i))).isAlive())
          connections.removeElementAt(i);
*/
     }
    catch(Exception e) {StreamManager.showError(e);}
    this.yield();
   }
  } // end run

 public void broadcast(String s)
 // effects:  outputs s to all PromptConnections
 {
  if (DEBUG) System.out.println ("broadcasting to promptframes: "+s);
  StreamManager out;
  Enumeration enum = connections.elements();
  while (enum.hasMoreElements())
    ((StreamManager)((PromptConnection)enum.nextElement()).output).send(s);
 } // end broadcast (String)

 public void broadcast (String s1, String s2)
 {broadcast (StreamManager.unparseMessage(s1,s2));}

 public void broadcast (String s1, String s2, String s3)
 {broadcast (StreamManager.unparseMessage(s1,s2,s3));}

 public void broadcast (String s1, String s2, String s3, String s4)
 {broadcast (StreamManager.unparseMessage(s1,s2,s3,s4));}

 public void broadcast (String s1, String s2, String s3, String s4, String s5)
 {broadcast (StreamManager.unparseMessage(s1,s2,s3,s4,s5));}

 public void broadcast (String s1, String s2, String s3, String s4, String s5, String s6)
 {broadcast (StreamManager.unparseMessage(s1,s2,s3,s4,s5,s6));}

 public void removeConn (PromptConnection c)
 //removes given connection from connections list
 {
  if (DEBUG) System.out.println ("removing promptconnection from list");
  if (!connections.removeElement(c))
    System.err.println ("Error: promptconnection not found in list");
 }
}

/*******************************************************************/
class PromptConnection extends ServerConnection
// This class encapsulates one broadcast socket connection
{
```

```
//CONSTANTS
  protected final static boolean DEBUG = PromptServer.DEBUG;

// FIELDS
  protected PSInLoop input_loop;
  private PromptServer ps;

// METHODS

  public PromptConnection(Socket s, PromptServer b, Database d)
  {
    super(s,d);
    System.out.println("s:" + s.toString());
//    System.out.println("input:" + input.toString());
//    System.out.println("output:" + output.toString());
    ps = b;
    input_loop = new PSInLoop(this, b, d);
  } // end PromptConnection constructor

  protected void doServerWork()
  // effects:  starts input loop
/* the following comments are obsolete
  // returns the Exception thrown, if any.
  // this exception isn't rethrown because the compile won't let it.
  // if it tries to throw an exception, it says Invalid Exception thrown
  // but if it doesn't throw it, it says, Exception Must Be declared in
  //    throws clause
*/
  {
    try {
      input_loop.start();
      input_loop.join();
//      input_loop.run(this);
    } catch (Exception e) {
      if (!(e instanceof SocketException)) StreamManager.showError(e);
    }
    dprint ("done with PromptConnection");
  } // end doServerWork

  protected void handleSocketException()
  //called if connection is lost with client
  {
    dprint ("Handling socket exception in PromptServer");

    //if user hasn't logged in yet, simply close connections
    if (!database.removeStreams(this))
      //if he has logged in, remove him from the database
      //actually, presumably, the client applet should send a DONESTRING,
      //even if the user closes his browser before hitting "Exit," but this
      //extra destroyPerson() is here just in case that DONESTRING is not
    //sent.  If it turns out that the DONESTRING was sent, nothing happens
    //anyway on the second call to destroyPerson()
    database.destroyPerson(input_loop.get_name());
  }  //handleSocketException

     public void destroy()
     {
      System.out.println ("PromptConnection.destroy() called");
      ps.removeConn(this);
//       input_loop.stop();
      try {
       mysocket.close();
      } catch (IOException e) { }
//          System.out.println ("IOException thrown");
//        }

//       super.destroy();
     }

  //PRIVATE METHODS
  private void dprint (String msg) {
   //prints to primary output if DEBUG flag is on
   if (DEBUG) System.out.println (msg);
  }
}          //PromptConnection

/*********************************************************************/
// This class constantly checks to see if there are any characters in the
// broadcast input stream
class PSInLoop extends InputLoop
{
//CONSTANTS
  protected final static boolean DEBUG = true;

//FIELDS
  PromptServer bc;
  static Database database;
  PromptConnection conn;
  String name;     //name of person connected to this InputLoop
  StreamManager output;
  int count = 0;    //used by displayRunning

// METHODS

  public PSInLoop(PromptConnection c, PromptServer b, Database d)
  {
   super(c.input);
   conn = c;
   bc = b;
   database = d;
  output = c.output;
} // end PSInLoop constructor

public String get_name() {return name;}

public boolean process_message(String client_message)
// effects:  broadcasts the input string to all the broadcast sockets
{
  dprint("processing (prompt):" + client_message);
  StringTokenizer tok = StreamManager.parseMessage (client_message);

  String code = tok.nextToken();
  dprint ("code: "+code);
  try {
    //FOR NEW USERS
    if (code.equals(sc.NEWUSERSTRING)) {
      //user just enterred page on browser
      //checks if there is space for more people
      if (database.atCapacity())
        output.send(sc.PERSONCAPSTRING);
      else {
        output.send (sc.GETNAMESTRING,database.getDiscs(),
                database.getPeople());
        database.addPConn ((PromptConnection)conn);
      }  //database.atCapacity()
      return true;
    }    //if code.equals NEWUSERSTRING

    if (code.equals(sc.USERNAMESTRING)) {
      //user has enterred his/her name
      //checks whether name is already in Database (or on disk)
      name = tok.nextToken();
      String pwd;

      if ((pwd = database.loadPassword(name))==null)
//        if (database.similarName(name)) output.send (sc.SIMNAMESTRING);
              output.send (sc.NEWNAMESTRING);
      else if (database.isActive(name)) output.send(sc.ACTIVENAMESTRING);
        else output.send (sc.GETPWDSTRING,pwd);
      return true;
    }    //if code.equals USERNAMESTRING

    if (code.equals(sc.CREATEPERSTRING)) {
       database.createPerson (name, tok.nextToken(),(PromptConnection)conn);
       return true;
    }    //if code.equals CREATEPERSTRING

    //DISCUSSIONS
    if (code.equals(sc.NEWDISCSTRING)) {
      //create new subdiscussion
      database.addDiscussion(tok.nextToken(),    //creater of discussion
                                   tok.nextToken(), //topic
                                   tok.nextToken(), //description
                                   tok.nextToken(), //parent discussion
                                   Integer.parseInt(tok.nextToken()), //capacity
                                   output);
   return true;
  }  //if code.equals NEWDISCSTRING

  if (code.equals(sc.CHANGEDISC)) {
   //changes appropriate person to appropriate discussion
   database.movePerson (name,  //of user
                          tok.nextToken());//of new discussion
   return true;
  } //if code.equals CHANGEDISC

  //IMPORTS
  if (code.equals(sc.PICWINSTRING)) {
   //new PictureWindow requested
   database.addImage (tok.nextToken(),    //user who imported it
          tok.nextToken(),            //discussion of import
          tok.nextToken(),            //title
                                   tok.nextToken());          //url
   return true;
  }  //if code.equals PICWINSTRING

  if (code.equals(sc.OBJSTRING)) {
   //new PictureWindow requested
   database.add3DObj (tok.nextToken(),    //user who imported it
          tok.nextToken(),            //discussion of import
          tok.nextToken(),            //title
                                   tok.nextToken());          //url
   return true;
  }  //if code.equals OBJSTRING


  if (code.equals(sc.COMMENTSTRING)) {
   //text to be broadcast on TalkWindows from SendCommWindow
   database.sendComment (tok.nextToken(),tok.nextToken());
   return true;
  }   //if code.equals COMMENTSTRING

  if (code.equals(sc.REMOVEIMPORT)) {
      //comment, picture, etc. to be removed
   database.removeImport (name,tok.nextToken(), //import type
                                   tok.nextToken());//id of import
   return true;
  }               //code.equals REMOVEIMPORT

  //WINDOWS
```

```
    if (code.equals(sc.CHATSTRING)) {
      //new SendChatWindow requested
      //client_request should include name of person who requested window
        database.addSlot(tok.nextToken());    //speaker
        return true;
    }    //if code.equals CHATSTRING

    if (code.equals(sc.CLOSECHAT)) {
      //someone closed his SendChatWindow
      database.removeSlot (tok.nextToken());
      return true;
    }   //if code.equals CLOSECHAT

    if (code.equals(sc.HISTWINSTRING)) {
          //new HistWindow requested
          database.addHistWin (tok.nextToken(),  //person who requested it
                         tok.nextToken(),    //discussion of the history
                         tok.nextToken());  //person of the history
            return true;
    }  //if code.equals HISTWINSTRING

    if (code.equals(sc.SETTEXT)) {
      //text to be broadcast on ChatWindows from SendChatWindow
      String spkr = tok.nextToken();
      String text = tok.nextToken();

      database.setChat (spkr,text);
      return true;
    }   //if code.equals SETTEXT

   //OTHER
    if (code.equals(sc.DONESTRING)) {
      //user specified is done
      if (!database.destroyPerson(name)) {
        //there was no username sent
        //user hasn't logged in yet, but wants to quit
        database.removeStreams((PromptConnection)conn);
      }      //if database.destroyPerson

          go = false;                    //end InputLoop
//        stop();             //this input loop is no longer needed
      return true;
    }    //if code.equals DONESTRING

   } catch (Exception e) {StreamManager.showError(e);}
   System.out.println ("Message not processed: "+client_message);
   return false;
  } // end process_message

  protected void handleSocketException() {
    conn.handleSocketException();
//   super.handleSocketException();
   }

  protected void displayRunning()
  //temporary
  //used to see whether input loops are hogging processor (when there is
  //        only one client logged in)
  {
//   if (count==0) System.out.println ("PSInLoop waiting for message");
//   else
         if (++count>10000) count = 0;
  }

  //PRIVATE METHODS
  private void dprint (String msg) {
    //prints to primary output if DEBUG flag is on
    if (DEBUG) System.out.println (msg);
  }
} // end PSInLoop
```

---

```
/*Recorder.java
contains Recorder,RecVector, & Record classes
Discussion and Person are subclasses of Recorder.  Anything that can have
a history is a Recorder.  A record is one instance of an object import,
a discussion creation, or comment posting.  A record contains the person
who performed the action, the discussion in which it was performed, and
important attributes of the action.  A RecVector is a vector of Records.
Each Recorder has a RecVector that contains the history of the Recorder's
actions.
*/

import java.util.*;
import java.io.*;
import StringCode;
import PromptServer;
import StreamManager;
import ImportInfo;

abstract class Recorder {
//CONSTANTS
  public final static boolean DEBUG = Database.DEBUG;
  public final static StringCode sc = Database.sc;
  public final static int WRITETOEND = -1;  //When HistLine has this value,
       //next time anything writes to the history file, it should
       //append to the end of the file

//FIELDS
  protected RecVector History = new RecVector(5,2);
  protected String ShortName;  //used for determining names of history files
  protected int HistLine = WRITETOEND;
//  protected int InitHistLen;  //length of vector when history is loaded
  protected int commId=0;  //id number of next comment
                        //first comment has id 0, next one is 1, etc.

//CONSTRUCTORS
  protected Recorder (String name)
  //determines ShortName name by eliminating all spaces from String name
  //thi
  {
   dprint ("Constructing Recorder: "+name);
   ShortName = StreamManager.findShortName(name);
   dprint ("ShortName: "+ShortName);
  }          //Recorder constructor

  public String loadHistory()
  //used for loading history for people or temporary recorders
  {
   return loadHistory ((Hashtable)null, (Hashtable)null, (Hashtable)null,
                                 (Vector)null, (PromptServer)null);
  }

  public String loadHistory(Hashtable imagelist, Hashtable objects,
          Hashtable commentlist, Vector subdiscs,PromptServer bc)
  //loads comments, objects, and subdiscussions from disk
  //returns null if history not loaded
  //otherwise, returns last discussion visited
  {
   dprint ("loading history for "+ShortName);
   String returnstr = null;
   boolean nodone = true; //whether the history file ends with
                        //a DONESTRING record
   try {
    RandomAccessFile inputfile=new RandomAccessFile (HistFile(),"r");
    String rectype;
    String user;
    String disc=null;
    String timestamp;
    String quoteln;
    StringBuffer quote;

    try {
     while (inputfile.getFilePointer() < inputfile.length() && nodone) {
          rectype = readInput(inputfile);
          if (rectype.equals(sc.COMMENTSTRING)||rectype.equals(sc.CHATSTRING))
          {
       quote = new StringBuffer();
       user = readInput(inputfile); //also includes id #
       disc = readInput(inputfile);
```

```
timestamp = readInput(inputfile);
if (!(quoteln = readInput(inputfile)).equals(sc.HISTDIVIDER)) {
  quote.append (quoteln);
  while (!(quoteln = readInput(inputfile)).equals(sc.HISTDIVIDER))
    quote.append ('\n'+quoteln);
}      //if quoteln
History.addElement(new Record (rectype,user,disc,
                                    timestamp,quote.toString()));
      dprint ("new comment/chat added:"+user);
      if (rectype.equals(sc.COMMENTSTRING))
        if (commentlist!=null) //if this is a Discussion
              commentlist.put (user,new ImportInfo(quote.toString()));
        else {
              commId++;
              dprint ("commId for incremented for "+ShortName+"; new value:"
                          +commId);
        }              //else
    }   //if rectype.equals COMMENTSTRING

    else if (rectype.equals(sc.PICWINSTRING)) {
      dprint ("adding image:"+ShortName);
skipInput(inputfile,3);  //skipping over user & discussion & time
      if (imagelist==null)
              skipInput (inputfile,2);  //this recorder is a user
      else imagelist.put (readInput(inputfile),   //title
                          new ImportInfo(readInput(inputfile)));        //url
      readInput(inputfile);  //skip histdivider
    }   //if rectype.equals PICWINSTRING

    else if (rectype.equals(sc.OBJSTRING)) {
      dprint ("Adding object:"+ShortName);
skipInput(inputfile,3);  //skipping over user & discussion & time
      if (objects==null)
              skipInput (inputfile,2);  //this recorder is a Person
      else objects.put (readInput(inputfile),       //title
                          new ImportInfo(readInput(inputfile)));        //url
      readInput(inputfile);  //skip histdivider
    }   //if rectype.equals OBJSTRING

    else if (rectype.equals(sc.NEWDISCSTRING)) {
      dprint ("Adding subdiscussion to "+ShortName);
      readInput(inputfile);  //skip over user
      int max=0;
      if (subdiscs==null) skipInput (inputfile,3);//this is a Person
else {
  disc = readInput(inputfile);
        readInput(inputfile);  //skip over timestamp
        max = Integer.parseInt(readInput(inputfile));
      }  //if subdiscs else
```

```
          quote = new StringBuffer();  //quote, in this case = description
          while (!(quoteln=readInput(inputfile)).equals(sc.HISTDIVIDER))
            quote.append(quoteln);
            if (subdiscs != null)
              subdiscs.addElement (new Discussion(disc, quote.toString(), bc,
                                   (Discussion)this, max));
        }   //if rectype.equals NEWDISCSTRING

        else if (rectype.equals(sc.REMOVEIMPORT)) {
          //Import removed; traverse through lists and remove it
          if (imagelist!=null) {
            //then this Recorder is a discussion
            skipInput(inputfile,3); //skip over remover's name, discussion,
                          //and timestamp
            String removed = readInput(inputfile);
            dprint ("removing import:"+removed);

            //remove removed from all three Hashtables
                //on client side, there's a guarantee that no two imports
                //have the same title, and that identifiers for Comments
                //use sc.IDDIVIDERs anyway, which will be untypable in the
                //final version.  Since every comment has a different ID
                //number for the same speaker, all comments are distinct.
                //thus, calling remove() on all three Hashtables will not cause
                //a removal any object that shouldn't be removed
            commentlist.remove(removed);
            imagelist.remove(removed);
            objects.remove(removed);
            readInput(inputfile);   //skip over HISTDIVIDER
          }                 //if imagelist != null
          else              //this is a Person
            skipInput(inputfile,5); //skip over everything
        }                   //if rectype.equals REMOVEIMPORT
        else if (rectype.equals(sc.DONESTRING)) {
          nodone = false;
          returnstr = inputfile.readLine(); //don't call readInput
                          //b/c we don't want to increment counter
    }  //if rectype.equals DONESTRING
    else {
          dprint ("Error: Invalid rectype:"+rectype);
          System.exit(1);
    }   //if rectype else
   }        //while inputfile
   inputfile.close();
  } catch (IOException e) {
   System.err.println ("Problem reading from history file.");
      inputfile.close();
   return null;
  }  //try/catch
} catch (IOException e) {
   System.err.println ("Problem opening history file for reading.");
   return null;
  } catch (Exception e) {
   StreamManager.showError(e);
   System.exit(1);
  }  //try/catch
  if (imagelist!=null ||        //this Recorder is A Discussion
   nodone)                      //no DONESTRING record in this file
   HistLine=WRITETOEND;
  dprint ("HistLine: "+HistLine);
  return returnstr;
 }  //load

 public void addRecord (Record r)
 //writes record to disk & to memory
 //rectype = chat or comment
 {
  dprint ("adding Record to "+ShortName);

  History.addElement(r);
  History.saveLastElement(HistFile(),HistLine);
  HistLine = WRITETOEND;

  /*In the future, History will not be kept in memory.  Although the
  following line is rather puzzling, since we just added the record to
  the history, it is only temporary;  in the future, nothing will be added
  to memory anyway, and all records will be saved through this procedure.
  (Right now, there are 4 or 5 procedures, each for a different type of
  record.)
  */
  if (r.getType().equals(sc.REMOVEIMPORT)) History.removeElement(r);
  System.out.println ("Current size of history: "+History.size());
 }       //addRecord

 public boolean recordImage (String user, String disc,
         String title, String adr)
 {return recordImport (sc.PICWINSTRING,user,disc,title,adr);}

 public boolean record3DObj (String user, String disc,
         String title, String adr)
 {return recordImport (sc.OBJSTRING,user,disc,title,adr);}

 public boolean recordChild (String user, Discussion child)
 //records onto disk that a subdiscussion has been created
 //returns true if successfully written
 {
   dprint ("Writing subdisc creation for "+ShortName);

   try {
    RandomAccessFile outputfile = new RandomAccessFile (HistFile(),"rw");
    try {
     if (HistLine==WRITETOEND) outputfile.seek(outputfile.length());
        else {
         for (int i=0;i<HistLine;++i) outputfile.readLine();
         HistLine=WRITETOEND;
        } //if HistLine else
        outputfile.writeBytes(sc.NEWDISCSTRING+'\n');
     outputfile.writeBytes(user+'\n');
     outputfile.writeBytes(child.getTopic()+'\n');
     outputfile.writeBytes((new Date()).toString()+'\n');
     outputfile.writeBytes(Integer.toString(child.MaxPeople)+'\n');
     outputfile.writeBytes(child.getDescription()+'\n');
     outputfile.writeBytes(sc.HISTDIVIDER+'\n');
     outputfile.close();
     return true;
    } catch (IOException e) {
     System.err.println ("Problem writing to history file: "+HistFile());
        outputfile.close();
        return false;
    } //try/catch
   } catch (IOException e) {
    System.out.println ("Problem opening history file for writing.");
   } //try/catch
   return false;
  } //recordChild

//PROTECTED FUNCTIONS
 protected Enumeration getHistory() {return History.elements();}

 protected String HistFile()
 //name of the file to which to save the history
 {return ShortName+".history";}

 protected void dprint (String msg) {
  //prints to primary output if DEBUG flag is on
  if (DEBUG) System.out.println (msg);
 }

//PRIVATE FUNCTIONS
 private String readInput (RandomAccessFile f)
 throws IOException
 //returns next line in f, and increments HistLine
 {
  HistLine++;
//  System.out.println ("HistLine:"+HistLine);
  return f.readLine();
 }

 private void skipInput (RandomAccessFile f, int l)
 throws IOException
```

```
  //skips over l lines in f, and increments HistLine by l
  {
//   dprint ("skipping input: "+l);
    HistLine+=l;
    for (int i=0;i<l;++i) f.readLine();
  }

  private boolean recordImport (String header, String user, String disc,
          String title, String adr)
  //records onto disk that an image has been imported
  //returns true if successfully written
  {
    dprint ("Writing image opening for "+ShortName);

    try {
     RandomAccessFile outputfile = new RandomAccessFile (HistFile(),"rw");
     try {
      if (HistLine==WRITETOEND) outputfile.seek(outputfile.length());
          else {
           for (int i=0;i<HistLine;++i) outputfile.readLine();
           HistLine = WRITETOEND;
          }
          outputfile.writeBytes(header+'\n');
      outputfile.writeBytes(user+'\n');
      outputfile.writeBytes(disc+'\n');
      outputfile.writeBytes((new Date()).toString()+'\n');
      outputfile.writeBytes(title+'\n');
      outputfile.writeBytes(adr+'\n');
      outputfile.writeBytes(sc.HISTDIVIDER+'\n');
      outputfile.close();
      return true;
     } catch (IOException e) {
      System.err.println ("Problem writing to history file: "+HistFile());
          outputfile.close();
          return false;
     }  //try/catch
    } catch (IOException e) {
     System.out.println ("Problem opening history file for writing.");
    }  //try/catch
    return false;
  }  //recordImage
}   //Recorder

/****************************************************************/

class RecVector extends Vector {
//CONSTANTS
  private final static boolean DEBUG = Recorder.DEBUG;
  private final static int WRITETOEND = Recorder.WRITETOEND;
                    //see explanation in Recorder
  public final static StringCode sc = Database.sc;

//CONSTRUCTORS
  RecVector(int x, int y) {super(x,y);}

//FUNCTIONS
  public boolean saveLastElement(String filename, int linenum)
  //saves history to disk
  {
    if (DEBUG) System.out.println ("Saving history to "+filename);

    try {
      RandomAccessFile outputfile
        = new RandomAccessFile (filename,"rw");
      Record lastrec = (Record)lastElement();

      try {
        if (linenum==WRITETOEND) outputfile.seek(outputfile.length());
            else {
              for (int i=0;i<linenum;++i) outputfile.readLine();
              linenum=WRITETOEND;
            }
            outputfile.writeBytes(lastrec.RecType+'\n');
        outputfile.writeBytes(lastrec.getSpeaker()+'\n');
        outputfile.writeBytes(lastrec.getTopic()+'\n');
        outputfile.writeBytes(lastrec.getDateTime()+'\n');
        outputfile.writeBytes(lastrec.getQuote()+'\n');
        outputfile.writeBytes(sc.HISTDIVIDER+'\n');
        outputfile.close();
      } catch (IOException e) {
        System.err.println ("Problem writing to history file: "+filename);
            outputfile.close();
//          InitHistLen = size();
            return false;
      }  //try/catch
    } catch (IOException e) {
      System.out.println ("Problem opening history file for writing.");
      return false;
    }  //try/catch
    return true;
  }  //save
}  //RecVector

/*****************************************************************/

class Record {
//used to save what people say
//CONSTANTS
private final static boolean DEBUG = Database.DEBUG;
```

```
//FIELDS
  protected String RecType; //type of record -- chat or comment
  protected String Speaker; //person who said it
  protected String Topic; //topic of discussion
  protected String Subject; //subject field of comment
  protected String Quote; //what the person said
  protected Date TimeStamp; //when this record was created

//CONSTRUCTORS
public Record (String r, String p, String d, String q)
{RecType=r;Speaker=p;Topic=d;Quote=q;TimeStamp=new Date();}

public Record (String r, Person p, Discussion d, String q)
{
  RecType = r;
  Speaker = p.getName();
  Topic = d.getTopic();
  Quote = q;
  TimeStamp = new Date();
/*
  if (DEBUG) {
   System.out.println ("record created:");
   System.out.println ("speaker: "+Speaker);
   System.out.println ("topic: "+Topic);
   System.out.println ("quote: "+Quote);
   System.out.println ("timestamp: "+TimeStamp);
  }  //if DEBUG
*/
}        //Record constructor

/*
public Record (String r, Person p, Discussion d, String s, String q)
{
  RecType = r;
  Speaker = p.getName();
  Topic = d.getTopic();
  Quote = q;
  Subject = s;
  TimeStamp = new Date();
}        //Record constructor
*/

public Record (String r, String p, String d, String t, String q)
//used only when reloading histories from disk
{
  RecType = r;
  Speaker = p;
  Topic = d;
  Quote = q;
  TimeStamp = new Date(t);
}           //Record constructor from strings

//PUBLIC FUNCTIONS
 public String getType() {return RecType;}
 public String getSpeaker() {return Speaker;}
 public String getTopic() {return Topic;}
 public String getSubject() {return Subject;}
 public String getQuote() {return Quote;}
 public String getDateTime() {return TimeStamp.toString();}
} //Record class
```

---

```
/*
ServerConnection.java
contains only ServerConnection class.

This class is used to open connections on the server side
with client machines.
*/

import java.net.*;
import java.io.*;
import StreamManager;
import Database;

public abstract class ServerConnection extends Thread
{
 protected Socket        mysocket;
 protected Database database;
 public StreamManager    output;
 public DataInputStream    input;
// protected abstract InputLoop input_loop;

 public ServerConnection(Socket s, Database d)
  {
   System.out.println("ServerConnection:init");
   mysocket = s;
   database = d;
   try {
     output = new StreamManager(new PrintStream(mysocket.getOutputStream()));
     input = new DataInputStream(mysocket.getInputStream());
   } catch ( Exception e ) {StreamManager.showError(e);}
/* now in InputLoop
   setDaemon(true); // may not be necessary
   setPriority(Thread.MIN_PRIORITY); // this should be fine, since input can
*/
                    //accumulate in the buffer without any problems
  } //constructor

      protected abstract void doServerWork();
      protected abstract void handleSocketException(); //used if clients
            //close browser without hitting "Exit" button

      public InetAddress getInetAddress()
      {return mysocket.getInetAddress();}

      public void run()
      {
        System.out.println("Connected to: " + mysocket.getInetAddress() +":"+ mysocket.getPort());
        doServerWork();
/*
        if (e!=null) {
          if (e instanceof SocketException) {
            //user closed browser
            System.out.println ("SocketException!");
            handleSocketException();
           }
          else {StreamManager.showError(e);}
        }  //if e!=null
*/
//            System.out.println ("Done with ServerConnection");
      }

   //destroy() should be overridden by subclasses to destroy InputLoop and
   //close socket
}
```

---

```
/* TalkSlot.java
includes only class TalkSlot

This class is used to keep track of one chat window (see TalkWindow.java)
of one user (see Person.java) on the server side.
*/
class TalkSlot {
//CONSTANTS
  public final static boolean DEBUG=Database.DEBUG;

//FIELDS
  private String Speaker;  //person currently using this slot
  private String Text=StringCode.EMPTYFIELD;

//CONSTRUCTORS
  public TalkSlot(String p) {
    if (DEBUG) System.out.println ("TalkSlot with text from "+p);
    Speaker=p;
  }        //TalkSlot (Person, String)

//PUBLIC FUNCTIONS
  public String getSpeaker() {return Speaker;}
  public String getText() {return Text;}

  public void setText(String s) {Text = s;}
}
```

## II. Prompting Frame

```
/*ButtonBar.java
contains ButtonBar class.

This is used to contain a set of Buttons (java.awt package)
*/

package WinPack;
import java.awt.*;
import java.util.*;
import java.net.*;
//import Display;

class ButtonBar extends Panel {
//a horizontal bar of buttons
public final static boolean DEBUG = Display.DEBUG;
  public ButtonBar() {
    setLayout(new FlowLayout(FlowLayout.LEFT));
    setForeground (Color.black);  //buttons are gray, though
    setBackground (Color.black);
  }     //ButtonBar constructor

  public Dimension size()
  {
    int numbuttons = countComponents();
    int width = 0;
    int ht = 15;
    Dimension d;

    for (int i=0;i<numbuttons;i++) {
      d = getComponent (i).preferredSize();
      width += d.width;
      if (d.height>ht) ht = d.height;
    }
    if (DEBUG) System.out.println ("Button bar dimensions:"+width+','+
                                              (ht+Display.MARGIN*2));
    return new Dimension (width, ht+Display.MARGIN*2);
  } //size()
}  //ButtonBar
```

---

```
/* CommentTitleBar.java
contains CommentTitleBar class
```

```
This class is the title bar for TalkWindows (ChatWindows too, although the
name may be deceiving.  It is just like a Title Bar, But the text is
left-justified.)
*/

package WinPack;
import java.awt.*;
import java.util.*;
import java.net.*;
//import Display;
//import ConfFrame;

class CommentTitleBar extends Canvas
{
// CONSTANTS
  public static final boolean DEBUG = Display.DEBUG;
// FIELDS
  public String title;

// METHODS
  public CommentTitleBar(String t)
  {
    title = t;
    setFont(ConfFrame.titleFont);
    setForeground(Color.white);
    setBackground(Color.black);
    resize (0,18);
  } // end CommentTitleBar constructor

  public void paint(Graphics g)
  // effects: draws titlebar with
  {
    int width = size().width;
    Dimension titledim = stringSize();
    g.setColor(Color.gray);
    g.fillRect(0,0,width,1);
    g.setColor(new Color(200,200,200));
    g.drawString(title, 0,(titledim.height-Display.MARGIN*2)-1);
    g.setColor(Color.gray);
    g.fillRect(0,17,width,1);

  } // end paint

  public void setText (String text)
  {
   title = text;
   repaint();
  }
public Dimension stringSize()
//returns size of title string, with margins
{
 FontMetrics f = getGraphics().getFontMetrics();
 if (DEBUG) System.out.println ("title size("+title+"):"+
                              (f.stringWidth (title)+2*Display.MARGIN)+','+
                              (f.getHeight()+2*Display.MARGIN));
 return new Dimension (f.stringWidth (title)+2*Display.MARGIN,
                              f.getHeight()+2*Display.MARGIN);
}

public boolean handleEvent(Event evt)
{
 switch(evt.id)
 {
  case Event.MOUSE_DOWN:
  {
   Container par = getParent();
       try {
    ((Desktop)(par.getParent())).select(par, evt.x, evt.y);
   } catch (ClassCastException e) { }
   return true;
  }
  default:
       return false;
 } //switch evt.id
} // end handleEvent

} // end CommentTitleBar
```

```
/*ConfFrame.java
Contains only ConfFrame class.

ConfFrame provides a control panel for the user. This
panel allows the user to create subdiscussions, post comments, and import
objects.
In the future, it may allow users to chat with other users and bring up
history windows to display the history of a discussion or user.
*/

package WinPack;
import java.awt.*;
import java.applet.Applet;
import java.util.*;
import java.io.PrintStream;
import java.net.*;
import StringCode;
import PromptFrame;
import StreamManager;
```

```
//public class ConfFrame extends Panel implements ImageLoader
public class ConfFrame extends Frame implements ImageLoader
{

// CONSTANTS
  protected static final boolean DEBUG = PromptFrame.DEBUG;
  protected static final StringCode sc = PromptFrame.sc;  //DK 7/18
  protected static final String CURDISCHEADER = "Discussion: ";

// FIELDS
  Applet app;
  public StreamManager output;

  //lists of stuff (used by some PromptWindows to display Choices)
  Hashtable Images = new Hashtable (1);
  Hashtable vrmlObjs = new Hashtable (1);
  Hashtable Comments = new Hashtable (3); //the values of this
          //Hashtable are actually never used; it's just made
          //into a Hashtable to let this program use
          //findImportList
  Vector DiscList = new Vector (4,2);
  Vector PersList = new Vector (6,2);

  //awt stuff
  public Desktop d;
  TitleBar title;
  int dwidth;  //Desktop width & height
  int dheight;
  protected static Font standardFont = new Font("Helvetica", Font.PLAIN, 10);
  protected static Font titleFont = new Font("TitleFont", Font.BOLD, 14);
  int TwWidth = 300;  //width of talkwindow display
  int TwHeight = 150;  //height
  int HwWidth = 100;
  int HwHeight = 200;  //for HistWindows

  //menu stuff
/* now uses Menubar
  ButtonBar menubar = new ButtonBar();
  Button NewChat = new Button("Chat");
  Button NewComm = new Button ("Post Comment");
  Button Exit = new Button ("Exit");
  Button HistButton = new Button ("History");
  Button SubDisc = new Button ("Create subdiscussion");
  Button PicButton = new Button ("Import Image");
  Button ObjButton = new Button ("Import VRML");
  Button RemoveButton = new Button ("Remove Import");
  DiscChoice SubDiscChoice = new DiscChoice(this);
*/

  MenuBar menubar = new MenuBar();
```

```
Menu fileMenu = new Menu("File");
Menu commMenu = new Menu("Comments");
Menu importMenu = new Menu ("Imports");
DiscChoice discMenu;

//in fileMenu
MenuItem HistButton = new MenuItem ("View history");
MenuItem Exit = new MenuItem ("Exit DDS");

//in commMenu
MenuItem NewChat = new MenuItem ("Chat");
MenuItem NewComm = new MenuItem ("Post a Comment");
MenuItem RemoveComm = new MenuItem ("Remove a Comment");

//in importMenu
MenuItem PicButton = new MenuItem ("Import Image");
MenuItem ObjButton = new MenuItem ("Import VRML object");
MenuItem RemovePic = new MenuItem ("Remove Import");
MenuItem Remove3D = new MenuItem ("Remove VRML object");

//in discMenu
MenuItem SubDisc = new MenuItem ("Create Subdiscussion");
MenuItem RemoveDisc = new MenuItem ("Remove Subdiscussion");
//other items added and removed dynamically as discussions
//    change

//info about state of ConfFrame
String Disc;
String User;
boolean inDatabase = false;

// CONSTRUCTORS
public ConfFrame(String disc, String user, StreamManager bc_output,
        int width, int height, PromptFrame client, String hostname)
{
  super("Design Discussion Space");
  //assigning values to fields
  System.out.println("ConfFrame:init");
  User = user;
  app = (Applet) client;
  setFont(standardFont);
  output = bc_output;
  dwidth = width;
  dheight = height;
  discMenu = new DiscChoice(output);

  //general layout of ConfFrame
  setLayout(new BorderLayout());
  setBackground(Color.black);
  setForeground(Color.white);
```

```
    //creating menubar
/*menubar now is a MenuBar
    menubar.setBackground(Color.lightGray);
    menubar.setForeground(Color.black);
    menubar.add(Exit);
*/

/* NOW ADDED IN setInDatabase
    menubar.add(NewComm);
    menubar.add(NewChat);
    menubar.add(PicButton);
    menubar.add(ObjButton);
    menubar.add(RemoveButton);
    menubar.add(HistButton);
    menubar.add(SubDisc);
    menubar.add(SubDiscChoice);
*/

    //desktop
    d = new Desktop(dwidth, dheight);
    add("Center", d);


/* Since ConfFrame extends Frame now, no title bar is needed
    //top panel = title bar and menu bar
    Panel TopPanel = new Panel();
    TopPanel.setLayout(new GridLayout(2,1));
    TopPanel.add(title = new TitleBar (disc));
    title.resize(title.size().width/2,title.size().height);
    TopPanel.add(menubar);
    add ("North",TopPanel);
*/
//   add ("North",menubar);

    reset (disc);
    //stuff needed for ConfFrame that extends Frame
    //(as opposed to Panel)
    setResizable(true);
    setTitle ("Design Discussion Space");
    show();

//   paintAll(getGraphics());
    System.out.println("ConfFrame:done with init");
  } // end ConfFrame constructor

  public void reset (String disc)
  {
    Disc = disc;
//   title.setText (disc);
```

```
    //resetting SubDiscChoice, if the user is actually in a
    //discussion (i.e., inDatabase)
    if (inDatabase) {
//      if (SubDiscChoice != null)

/*now menubar is a MenuBar
        menubar.remove(SubDiscChoice);
      SubDiscChoice = new DiscChoice(this);
      SubDiscChoice.addItem (CURDISCHEADER+disc);

      //add DEFAULTDISC so that user can go back to the original
      //discussion at any time
      if (!disc.equals(sc.DEFAULTDISC)) SubDiscChoice.addItem(sc.DEFAULTDISC);

      //add to menubar
      menubar.add(SubDiscChoice);
      menubar.paintAll (menubar.getGraphics());
*/
      setDisc(disc);
    }  //if inDatabase

    Images = new Hashtable (1);
    vrmlObjs = new Hashtable (1);
    Comments = new Hashtable (3);
  } //reset

//PUBLIC METHODS
  public String getUser() {return User;}
  public String getDisc() {return Disc;}

  public void setUser(String u) {
    dprint ("Setting user to "+u);
    User = u;
  }

  public void setDisc (String d)
  //sets Disc and changes title bar appropriately
  {
    dprint ("setting discussion to "+d);
    Disc = d;
    setTitle ("Design Discussion Space -- "+Disc);
//    title.setText(d);

    //change SubDiscChoice appropriately
    discMenu.reset(d);
/*menubar now is a MenuBar
    menubar.remove(SubDiscChoice);
    SubDiscChoice = new DiscChoice(this);
    SubDiscChoice.addItem (CURDISCHEADER+d);
```

```
   if (!d.equals(sc.DEFAULTDISC)) SubDiscChoice.addItem(sc.DEFAULTDISC);
   menubar.add(SubDiscChoice);
   menubar.paintAll (menubar.getGraphics());
*/
 }     //setDisc

 public boolean handleEvent(Event evt)
 // effects:  specifies how to respond to NewChat and Exit button
 // presses.  If any buttons are added to the menubar then the code
 // to handle its button press should be added here.
 {
   if (evt.id == Event.ACTION_EVENT) {
     Object target= evt.target;

     if (target.equals(Exit)) {
       if (DEBUG) System.out.println("Exit Pressed");
              d.removeAllWindows();
       ((PromptFrame)getParent()).destroy();
       return true;
     }       //if target.equals Exit

     if (target.equals(NewChat)) {
       if (DEBUG) System.out.println("NewChat pressed");
       if (d.findWindow(sc.SENDCHATSTRING,sc.EMPTYFIELD)==null)
              output.send(sc.CHATSTRING,User);
       d.addWindow (new SendChatWindow(output));
       return true;
     }  //if target.equals NewChat

     if (target.equals(NewComm)) {
       if (DEBUG) System.out.println ("NewComm pressed");
       d.addWindow (new SendCommWindow(output));
       return true;
     }  //if target.equals(NewComm)

     if (target.equals(HistButton)) {
       if (DEBUG) System.out.println ("Discussion HistButton pressed");
       d.addWindow (new NewHistWin (DiscList,PersList,
                              output));
       return true;
     }     //if target.equals HistButton

     if (target.equals(PicButton)) {
       if (DEBUG) System.out.println ("PicButton pressed");
       d.addWindow (
          new NewPicWin(output, sc.PICWINSTRING));
       return true;
     }  //if target.equals PicButton

     if (target.equals(ObjButton)) {
```

```
    if (DEBUG) System.out.println ("ObjButton pressed");
    d.addWindow (
       new NewPicWin(output, sc.OBJSTRING));
    return true;
  }  //if target.equals ObjButton

  if (target.equals(RemoveComm)) {
    dprint ("Remove Comment pressed");
    d.addWindow (new RemovePrompt (sc.COMMENTSTRING,
          Comments.keys(), output));
    return true;
  } //target.equals RemoveComm

  if (target.equals(RemovePic)) {
    dprint ("Remove Image pressed");
    d.addWindow (new RemovePrompt (sc.PICWINSTRING,
          Images.keys(), output));
    return true;
  } //target.equals RemovePic

  if (target.equals(Remove3D)) {
    dprint ("Remove 3-D object pressed");
    d.addWindow (new RemovePrompt (sc.OBJSTRING,
          vrmlObjs.keys(), output));
    return true;
  } //target.equals Remove3D

  if (target.equals(SubDisc)) {
    if (DEBUG) System.out.println ("SubDisc pressed");
    d.addWindow (new NewDiscWin(output, DiscList));
    return true;
  }  //if target.equals SubDisc
 }   //if evt.id

 //maybe evt is a request to change discussion
 return discMenu.handleEvent(evt);
} // end handleEvent

public void setDiscAndPers (String discs, String people)
//sets DiscList and PersonList
//discs and people are string representations of the discussions & people
//all names are separated by sc.PARAMDIVIDER
{
 dprint ("setting DiscList: "+discs);
 DiscList = output.parseList (discs);

 dprint ("setting PersList: "+people);
 PersList = output.parseList (people);
}    //setDiscAndPers
```

```
public void setInDatabase ()
//user is now in the database
//therefore, menu buttons besides exit will appear
{
  if (inDatabase) return; //don't need to do anything
  inDatabase = true;  //so we won't do anything again

  dprint ("user is now in database; adding menu buttons");

  //add menus
  menubar.add(fileMenu);
  menubar.add(commMenu);
  menubar.add(importMenu);
  menubar.add(discMenu);

  //adding stuff to fileMenu
  fileMenu.add(HistButton);
  fileMenu.addSeparator();
  fileMenu.add(Exit);

  //for commMenu
  commMenu.add(NewChat);
  commMenu.add(NewComm);
  commMenu.addSeparator();
  commMenu.add(RemoveComm);

  //for importMenu
  importMenu.add(PicButton);
  importMenu.add(ObjButton);
  importMenu.addSeparator();
  importMenu.add(RemovePic);
  importMenu.add(Remove3D);

  //for discMenu
  discMenu.add(SubDisc);
  discMenu.add(RemoveDisc);
  discMenu.addSeparator();
 //other items added and removed dynamically as discussions
 //    change

  setMenuBar(menubar);

/*  Before, menubar was a Panel.  Now, it's a MenuBar
  menubar.add(NewComm);
  menubar.add(NewChat);
  menubar.add(PicButton);
  menubar.add(ObjButton);
  menubar.add(RemoveButton);
  menubar.add(HistButton);
  menubar.add(SubDisc);
```

```
   menubar.add(SubDiscChoice);
*/
  }

  public boolean setComments (String list)
  //unparses list to get new value for Comments
  //first, checks whether the size of Comments is zero
  //if it is, then it assigns Comments the new value and returns
  //   true
  //if not, does nothing and returns false
  {
   if (Comments.size()!=0) {
     System.err.println ("ERROR: trying to set Comments in"
       +"promptframe when Comments isn't empty");
     return false;
   }      //if Comments.size

   //check is OK; proceed
   Vector commVec = StreamManager.parseList(list);

   //convert to Hashtable
   for (Enumeration enum=commVec.elements();
      enum.hasMoreElements();
      Comments.put(enum.nextElement(),sc.EMPTYFIELD));
   return true;
  }    //setComments

  public void getName(String text)
  //text = text for window that prompts for name
  //there is room for this user
  //prompts user for his name
  //then sends name to server to check whether it has already been used
  {
   if (DEBUG) System.out.println ("Getting name of user");
   d.addWindow(new NameWindow(output, (PromptFrame)getParent(),
             text, PersList));
  }         //getName

  public void newPwd()
  {
   if (DEBUG) System.out.println ("Getting new password");
   d.addWindow (new NewPwdWindow (output));
  }

  public void askPwd(String pwd)
  {
   if (DEBUG) System.out.println ("Asking for password");
   d.addWindow (new AskPwdWindow (pwd, output));
  }
```

```
 public boolean addWindow(String winparams)
 {
   dprint ("adding window to ConfFrame:"+winparams);
   StringTokenizer tok = StreamManager.parseMessage (winparams);
   String code = tok.nextToken();

   if (code.equals(sc.CHATSTRING)) {
     d.addWindow (new ChatWindow(tok.nextToken(), tok.nextToken(),
                               Disc, TwWidth, TwHeight, output));
     return true;
   }  //if code.equals CHATSTRING

   if (code.equals(sc.HISTWINSTRING)) {
    d.addWindow (new HistWindow (tok.nextToken(),tok.nextToken(),
                                        tok.nextToken(), HwWidth, HwHeight));
     return true;
   }  //if code.equals HISTWINSTRING

   if (code.equals(sc.ERRWINSTRING)) {
     String title = tok.nextToken();
     String errWinparams = StreamManager.restOf(tok);
/*
     if (errWinparams.equals(sc.PERSONCAPSTRING)) {
       if (DEBUG) System.out.println ("Telling user server is at capacity");
       d.addWindow(new ErrorWindow (this, "Sorry!",
            new LabelSet("Sorry, but the conference is currently full",
                    "Please try again some other time.")));
     }  //if tok.nextToken()
*/
     d.addWindow (new ErrorWindow (title,
       new LabelSet(errWinparams)));
     return true;
   }  //if code.equals PERSONCAPSTRING

   System.err.println ("Error: invalid window type requested: "+code);
   return false;
 }         //addWindow

 public void addFinalWin (String title, String message)
 //creates an ErrorWindow that automatically closes ConfFrame
 //    when the user hits OK
 {
  d.addWindow (new ErrorWindow (title, new LabelSet(message),
      true));    //true indicates ConfFrame should be closed
 }

 public void removeWindow(DisplayWindow w)  //DK 7/14
 {d.removeWindow(w);}

 public void addComment (String disc, String id, String comm)
```

```
//someone has posted a comment comm in discussion disc
//id = speaker & idnumber
//updates history and adds to Comments list
{
 String spkr = output.parseCommIdGetSpkr (id);
 updateHistory (disc,id,comm);
 if (disc.equals(Disc)) Comments.put (id,sc.EMPTYFIELD);
}

public void addImage (String title, String url)
//a user in this discussion has imported something
{
 dprint ("adding image to promptframe: "+title+','+url);
 Images.put(title,url);
}    //addImage

public void add3DObj (String title, String url)
//a user in this discussion has imported something
{
 dprint ("adding vrml object to promptframe: "+title+','+url);
 vrmlObjs.put(title,url);
}    //add3DObj

public void removeComment (String disc, String id)
{
 dprint ("removing comment from prompttframe: "+id);
 if (disc.equals(Disc))  //if the comment is in this discussion
   if (Comments.remove(id)==null)
     //comment wasn't found
     System.err.println ("Error:trying to remove comment"
                 +"that doesn't exist");
 /*should also update history of removal of comment
   issues:  should it be removed from the histwindow, or
        should the removal simply be logged?
        If it's removed, how do we know which comment to
        remove?
  */
}

public void removeImport (String type, String id)
//removes pictures and 3-D objects (but not comments) from
//list of imports
//comments are taken care of by removeComm
{
 dprint ("removing import from promptframe: "+id);
 findImportList(type).remove (id);
}

protected boolean hasImport (String title, String url)
//checks whether there is an import with the same title OR url
```

```
 //used to make sure user doesn't import things with same title
 //    or url
 {return Images.containsKey(title)||
      vrmlObjs.containsKey(title)||
      Images.contains(url)||vrmlObjs.contains(url);}

 public void addPerson (String name)
 //another user has logged in for first time
 //update PersList
 {
   if (PersList.contains(name))
     System.err.println ("Error: trying to person to PersList twice:"+name);
   else {
     dprint ("adding Person to PersList: "+name);
     PersList.addElement(name);
   }
 }  //addPerson

 public void addSubDisc (String parent, String discname)
 //adds button of link to subdiscussion
 {
   dprint ("adding subdiscussion button: "+discname);
   dprint ("current disc: "+Disc);

         if (!DiscList.contains(discname)) DiscList.addElement(discname);
         if (parent.equals(Disc)) {
            discMenu.addSubDisc(discname);
/*
      SubDiscChoice.addItem(discname);
                     System.out.println("this name is being added to Choice:" + discname
                               + "  ");
*/
    }
  }

  public void addSuperDisc (String discname)
  //adds button of link to parent discussion
  {
//    dprint("adding superdiscussion button: "+discname);
     discMenu.addSuperDisc(discname);
//    if (!discname.equals(sc.DEFAULTDISC)) SubDiscChoice.addItem(discname);
//   menubar.paintAll(menubar.getGraphics());
  }

  public void closeTalkWin (String spkr)
  //closes ChatWindow of spkr
  {
   d.removeWindow(d.findWindow(sc.CHATSTRING,spkr));
  }
```

```
public void setText (String spkr, String disc, String text)
//sets text of appropriate ChatWindow
//updates appropriate HistWindows
{
 dprint ("setting Text: "+spkr+','+text);
 try {((ChatWindow)(d.findWindow(sc.CHATSTRING,spkr))).setText(text);}
 catch (NullPointerException e) {}
 updateHistory (spkr, disc, text);
}      //setText

public Image getImage(String image_url_st) {
 return app.getImage(app.getDocumentBase(), image_url_st);
}

public Image getImage(URL image_url) {
 return app.getImage(image_url);
}

public void updateHistory (String disc, String spkr, String text)
//updates appropriate history windows when a new comment is made
// either thru SendChatWindows or thru SendCommWindows
{
 /*THIS CODE RUNS RATHER INEFFICIENTLY -- RIGHT NOW, IT
 CALLS d.findWindow 3 TIMES; SO IT TRAVERSES THROUGH THE
 WINDOWS VECTOR 3 TIMES.  COULD WRITE SOMETHING SO IT
 TRAVERSES ONLY ONCE
 */
 dprint ("updating history");

 //look for HistWindow with no discussion specified, but
 //speaker = spkr
 HistWindow win = (HistWindow)d.findWindow (sc.HISTWINSTRING,
        HistWindow.unparseHistWinInfo(sc.EMPTYFIELD,spkr));
 //if that HistWindow is found, add an entry, with the
 //current time as the timestamp
        if (win!=null) win.appendText
              (disc+": "+text+"\n"+new Date()+"\n\n");

 //look for HistWindow with no speaker specified, but with
 //discussion = disc
 win = (HistWindow)d.findWindow (sc.HISTWINSTRING,
        HistWindow.unparseHistWinInfo(disc,sc.EMPTYFIELD));
        if (win!=null) win.appendText
              (spkr+": "+text+"\n"+new Date()+"\n\n");

 //look for HistWindow with both fields specified
 win = (HistWindow)d.findWindow (sc.HISTWINSTRING,
        HistWindow.unparseHistWinInfo(disc,spkr));
        if (win!=null) win.appendText
              (text+"\n"+new Date()+"\n\n");
 } //updateHistory

 //PRIVATE METHODS
private Hashtable findImportList (String list)
//returns Comments, Images, or vrmlObjs
{
 if (list.equals(sc.COMMENTSTRING)) return Comments;
 if (list.equals(sc.PICWINSTRING)) return Images;
 if (list.equals(sc.OBJSTRING)) return vrmlObjs;
 else System.err.println ("Error: Import list type not found:"
      +list);
 return null;
}


 private void dprint (String s)
 {if (DEBUG) System.out.println (s);}
}     //ConfFrame class
```

---

```
/*Desktop.java
contains Desktop class.

Desktop is a Panel that contains all interface windows in ConfFrame (see
ConfFrame.java). It uses ScaleLayout (see ScaleLayout.java), which allows
users to move the windows around on the desktop.
*/
package WinPack;
import java.awt.*;
import java.util.*;
import PromptFrame;
import StringCode;

class Desktop extends Panel implements ImageLoader
{
// CONSTANTS
 public static final boolean DEBUG = ConfFrame.DEBUG;
 public static final StringCode sc = ConfFrame.sc;
 private static final int INITSSWINSIZEX = 0; //initial dimensions of
 private static final int INITSSWINSIZEY = 0; //SelfSizedWins when they are
          //are first placed on Desktop. The windows are then resized
          //appropriately. This process is used because getGraphics() won't
          //work until a window has actually been placed on the desktop.
          //Therefore, the appropriate window size can not be calculated.


// FIELDS
 public ScaleLayout sl;
 private Component selected;
 private int mouseDownX; //x coordinate of when mouse button is pressed
 private int mouseDownY; //y coordinate
 private int slw;      //width & height of selected window
 private int slh;      //window is selected when user clicks on title
 public Dimension select_offset;
 int def_hx = 200;  //default x pos for new history windows
 int def_hy = 0;   //y pos
 int def_x = 20;   //default x position for new windows (except histories)
 int def_y = 20;   //y position
 int dcwidth = 300;  //default size for ChatWindows
 int dcheight = 160;
 int dswidth = 300;  //for SendWindows
 int dsheight = 160;
 int dhwidth = 300;  //for HistWindows
 int dhheight = 150;
 Vector EWindowList = new Vector (5,2);  //vector of open EmbeddedWindows

// METHODS
 public Desktop(int width, int height) {
  sl = new ScaleLayout(new Dimension(width, height));
  resize(width, height);
  setLayout(sl);
  /*should be changed to incorporate default window positions/sizes*/
 } // end Desktop constructor

 private void place(String name, Component c, int px, int py,
                               int w, int h)
 // effects: this method places c at location px, py on the scalelayout and resizes the component to w*d
 {
  System.out.println("In place");
  System.out.flush();
  add(name, c);
  System.out.println("just added");
  sl.setConstraint(c, px, py, w, h);  //also draws component
  System.out.println("just set constraints, Height = " + h);
 } // end place

 private void plop(String name, Component c, int px, int py, int w, int h)
 // effects: places c with specified arguments and also informs the ScaleLayout to reset constraints
 {
  if (DEBUG) System.out.println ("plopping window:"+px+','+py+','+w+','+h);
  place (name, c, px, py, w, h);
 } // end plop

 protected int getComponentIndex(Component c)
 // effects: returns the index of c in this Desktop container
 // returns -1 if c is not on this DeskTop
 {
  int numcomponents = countComponents();
  for (int i = 0; i < numcomponents; i++)
   if (c.equals(getComponent(i))) return i;
  return -1;
 } // end getComponentIndex

 protected void raise(Component c)
 // effects: is supposed to "raise" to lay on top of all other components in Desktop
 {
   System.out.println("raising window #" + getComponentIndex(c));

   remove(c);
   add (c);
 } // end raise

 public boolean select(Component c, int cx, int cy)
 // effects: next mouseUp event will cause c to be moved to the location of that event.
 // cx & cy are the coordinates relative to component
 {
   System.out.println("select!");
   selected = c;
   ScaleConstraint sc = sl.getConstraint(c);
   slw = sc.width;
   slh = sc.height;
   mouseDownX = sc.x;          //used later to detect whether user
   mouseDownY = sc.y;          //actually moves window
   System.out.println("selected:" + selected.toString());
   select_offset = new Dimension(cx, cy);
   raise(c);
   return true;
 } // end select

 public boolean mouseUp(Event evt, int x, int y)
 //used to move windows on desktop
 //sends message to user notifying that window was moved if it's a picture
        //or commentwindow
 {
//   System.out.println("mouseUp");
   if (selected != null)
   {
     int new_x = x-select_offset.width;  //DK 7/31
     int new_y = y-select_offset.height;  //DK 7/31

     if (mouseDownX != new_x || mouseDownY != new_y) {
     //window has actually moved
       sl.setConstraint (selected, new_x, new_y, slw, slh);
     }  //if MouseDownX ...
     selected.repaint();
     selected = null;
   }
   return true;
 } // end mouseUp

 public void select(Component c)
```

```
// effects: selects c with no offset
{
  selected = c;
} // end select

public Image getImage(String image_url_st) {
  return ((ImageLoader) getParent()).getImage(image_url_st);
}

public Image getImage(java.net.URL image_url) {
  return ((ImageLoader) getParent()).getImage(image_url);
}

public EmbeddedWindow findWindow (String wintype,String params)
//checks if any of the windows match the given parameters
//returns null if none do
{
  if (DEBUG) System.out.println ("Looking for window: "+wintype+','+params);
  EmbeddedWindow nextwin;
  for (Enumeration enum=EWindowList.elements(); enum.hasMoreElements();)
    if ((nextwin=(EmbeddedWindow)enum.nextElement()).matches(wintype,params))
      return nextwin;
  if (DEBUG) System.out.println ("Window not found.");
  return null;
}   //findWindow

public void addWindow (EmbeddedWindow w)
//uses default window sizes & positions
{
  if (DEBUG) System.out.println ("Adding window without paramaters");
  if (w instanceof SelfSizedWin) //pictures, errors, prompts, comments
    addWindow (w,def_x,def_y,INITSSWINSIZEX,INITSSWINSIZEY);  //use default values for pos &
dummy
                                            //values for size
  else              //senders, histories, chats
  if (w instanceof SendWindow)
    addWindow (w,def_x,def_y,dswidth,dsheight);
  else if (w instanceof HistWindow)
    addWindow (w,def_hx,def_hy,dhwidth,dhheight);
  else if (w instanceof ChatWindow)
    addWindow (w,def_x,def_y,dcwidth,dcheight);
} //addWindow (DisplayWindow)

public void addWindow (EmbeddedWindow win, int x, int y, int w, int h)
//adds win with given default coordinates & size (w=width; h=height)
{
  EWindowList.addElement(win);
  if (win instanceof SelfSizedWin) {
    if (DEBUG) System.out.println ("adding resizable window");
    SelfSizedWin sswin = (SelfSizedWin) win;
```

```
     plop ("", win, x, y, INITSSWINSIZEX, INITSSWINSIZEY);
     Dimension winsize = sswin.size();  //can't be called until plop is called
     sl.setConstraint ((Component)win, x, y, winsize.width, winsize.height);
    } //if win instanceof
    else plop("",win, x, y, w, h);
    win.paintAll(win.getGraphics());
  } //addWindow (DisplayWindow,int,int,int,int)

/* THIS IS USED BY GTT -- NOW ALL MOVEMENTS ARE DIRECTED TO ConfClient
CLASS
  public void moveWindow (String wintype, String win_id, int new_x, int new_y)
  //moves window on desktop to new position
  {
    if (DEBUG) System.out.println ("moving: "+win_id+"to "+new_x+","+new_y);
    EmbeddedWindow winmoved = findWindow (wintype,win_id);
    ScaleConstraint sc = sl.getConstraint(winmoved);
//   int wmw = sc.width;
//   int wmh = sc.height;

//   sl.setConstraint (winmoved, new_x, new_y, wmw, wmh);
    sl.setConstraint (winmoved,new_x,new_y);
    winmoved.repaint();
  } //moveWindow
*/

// public void setDefaultWinPos (int x, int y) {def_x = x; def_y = y;}

  public void removeWindow (DisplayWindow w) {
    if (DEBUG) System.out.println ("removing window from desktop");

    if (w.isEmbedded()) {
      EWindowList.removeElement(w);
      remove ((EmbeddedWindow)w);
    } //if w.isEmbedded

    if (DEBUG) System.out.println ("Window removed: "+w.Type());
  } //removeWindow

  public void removeAllWindows()
  //removes all windows from desktop
  {
    if (DEBUG) System.out.println ("Removing all windows:");

    while (!EWindowList.isEmpty())
      removeWindow ((DisplayWindow)EWindowList.firstElement());
  } //removeAllWindows

//PRIVATE METHODS
  private WinDefault getDefault (EmbeddedWindow w)
  //returns WinDefault of the given window, using the desktop's scalelayout
```

```
 {return new WinDefault (w.Type(),w.identifiers(),sl.getConstraint(w));}
} // end Desktop
```

---

```
/* DiscChoice.java
contains DiscChoice class.

This class is a subclass of java.awt.Menu.  It handles events for
switching between discussions.  Note that it contains two MenuItems
that it does not handle: one for adding subdiscussions, and one for
removing.  When an ConfFrame.handleEvent() is invoked, it checks
whether it can handle the event.  The only events in the menu bar
that it does not handle are those that cause switches between
discussions.  In that case, DiscChoice.handleEvent() is called.
*/


package WinPack;
import java.awt.*;
import java.util.*;
import StringCode;
import StreamManager;

class DiscChoice extends Menu {
//CONSTANTS
public static final boolean DEBUG = ConfFrame.DEBUG;
public static final StringCode sc = ConfFrame.sc;

//FIELDS
Hashtable discs = new Hashtable(2);  //hashes MenuItems to
        //the titles of their corresponding discussions
StreamManager output;
boolean noSubdiscs=true;   //whether there are no subdiscs
                //in discs.

//CONSTRUCTOR
public DiscChoice (StreamManager o)
//simply puts title on Menu
{
 super("Discussions");
 output = o;
}

//PUBLIC METHODS
public void reset(String disc)
//called when the user changes discussions
//removes all MenuItems in discs
{
 dprint ("resetting DiscChoice");
 //first, get rid of all the MenuItems
 for (Enumeration discButtons = discs.keys();
      discButtons.hasMoreElements();)
   remove((MenuItem)discButtons.nextElement());

 discs = new Hashtable(2);
 noSubdiscs = true;


 //we want the user to be able to go back to the default
 //discussion at any time; so add that, unless he's already
 //in the default disc
 if (!disc.equals(sc.DEFAULTDISC)) {
   MenuItem def = new MenuItem ("To the top:"+sc.DEFAULTDISC);
   discs.put (def,sc.DEFAULTDISC);
   add(def);
 }    //if !disc
}

public boolean handleEvent (Event evt)
//handles events in ConfFrame that ConfFrame can't handle
//specifically, notifies server when user wants to change
// discussions
{
// dprint ("handling event in DiscChoice");

  String newdisc = (String)discs.get(evt.target);
  if (newdisc==null) return false; //evt couldn't be handled

  //otherwise, send a message
  dprint ("Discussion Selected: "+newdisc);
  output.send(sc.CHANGEDISC,newdisc);
  return true;
}

public void addSuperDisc (String disc)
//adds superdiscussion named disc to Menu, unless it is the
//default discussion (because the default discussion has already
//been added to the Menu.
{
  dprint ("adding superdiscussion to DiscChocie: "+disc);
  if (!disc.equals(sc.DEFAULTDISC)) {
    MenuItem newdisc = new MenuItem ("Up To "+disc);
    discs.put (newdisc,disc);
    add(newdisc);
  }    //if !disc
}      //addSuperDisc

public void addSubDisc (String disc)
{
  dprint ("adding subdiscussion to DiscChoice: "+disc);
  if (noSubdiscs)
   noSubdiscs = false;

  MenuItem newdisc = new MenuItem ("Go To "+disc);
  discs.put (newdisc,disc);
  add(newdisc);
}      //addSubdisc

//PRIVATE METHODS
  private void dprint (String s)
  {if (DEBUG) System.out.println (s);}
}      //DiscChoice

/* OLD VERSION
before, DiscChoice extended Choice.  At that time, menubar was
a Panel, as opposed to a MenuBar.  All the menu items were buttons.
Now, they are MenuItems.
/************************************************************************** /
//RONMAC ADDED THIS CLASS TO SINGLE DISCUSSION BUTTONS TO DISCUSSION CHOICE
class DiscChoice extends Choice {  //DK 7/16
//This class used to extend Choice, until the menus were incorporated
//CONSTANTS
  private final static StringCode sc = ConfFrame.sc;
  private final static boolean DEBUG = ConfFrame.DEBUG;

//FIELDS
  String Name;  //of corresponding discussion
  ConfFrame frame;

//CONSTRUCTORS
  DiscChoice (ConfFrame f) {
   //super (prefix+name);
   //Name = name;
   frame = f;
  }

//METHODS
  public boolean action(Event e, Object o)
  //sets Disc to new discussion
  //sends message to ConferenceServer
  {
   String Name = (String) o;
   if (DEBUG) System.out.println ("Discussion Selected: "+Name);
//   if (!Name.equals (ConfFrame.DISCCHOICETITLE))
              //if he actually chose something
//   frame.broadcast_output.send(sc.CHANGEDISC,frame.User,Name);
   if (!Name.startsWith(ConfFrame.CURDISCHEADER))
    frame.output.send(sc.CHANGEDISC,Name);
   return true;
```

```
 }      //takeAction
}  //DiscChoice
*/
```

---

```
/* Display.java
Contains Display class.

Display is a TextArea that is used by the windows of the Desktop.
It also contains some static functions for calculating dimensions,
especially when resizing SelfSizedWins (see DisplayWindow.java)
*/

package WinPack;
import java.awt.*;
import java.util.*;
import java.net.*;

/*
        This is the actual widget where the text is displayed.
        Note that this was intended to be an all-purpose widget
        that could display both text and images.  Thus Display should
        be completely rewritten at some point.
*/

class Display extends TextArea {

// CONSTANTS
  public static final boolean DEBUG = true;
  protected static int MARGIN = 1;  //between labels, prompts, etc.
  protected static int TEXTMARGIN = 35;  //for text fields & text areas

// FIELDS
  String back; // url for background.  does not work.

// METHODS
  public Display () { };

  public Display(String text, int width, int height)
  {
    if (DEBUG) System.out.println("creating display!");
    resize (width,height);
    setBackground(Color.black);
    setForeground(Color.white);
    setText (text);
    if (DEBUG) System.out.println("Done creating display");
  } // end Display constructor

  public Display(int width, int height, String background) {
    back = background;
    if (DEBUG) System.out.println("creating display!");
    resize (width,height);
    setBackground(Color.black);
    setForeground(Color.white);
    if (DEBUG) System.out.println("Done creating display");
  } // end Display constructor

  protected void insertString(String s)
  {
    appendText(s);
  } // end insertString

  public void paint(Graphics g) {
  // effects: is supposed to paint image in background of Display
/*   try
    {
      URL testurl = new URL(back);
      System.out.println(testurl.toString());
      Image letters = this.getImage(testurl);
      System.out.println(letters.toString());
      System.out.println("created image");
      g.drawImage(letters, 0, 0, this);
    }
    catch(Exception e)
    {
        System.err.println("Exception:");
        e.printStackTrace();
    }
*/
  } // end paint

//STATIC FUNCTIONS
  public static int max (int a, int b)
  {return a>b?a:b;}

  public static int max (int a, int b, int c)
  {
    int r = a>b?a:b;
    return r>c?r:c;
  }

  public static int max (int a, int b, int c, int d)
  {
    int r = a>b?a:b;
    if (c>r) r = c;
    return d>r?d:r;
  }

  public static int max (int a, int b, int c, int d, int e)
  {
    int r = a>b?a:b;
  if (c>r) r = c;
  if (d>r) r =d;
  return e>r?e:r;
 }

 public static int max (int a, int b, int c, int d, int e, int f)
 {
  int r = a>b?a:b;
  if (c>r) r = c;
  if (d>r) r =d;
  if (e>r) r=d;
  return f>r?f:r;
 }

 public static Dimension sizeDims (Dimension a, Dimension b)
 //given a & b, which are dimensions of components of a SelfSizedWin,
 //determines dimensions of that window
 {return new Dimension (max(a.width, b.width),a.height+b.height);}

 public static Dimension sizeDims (Dimension a, Dimension b, Dimension c)
 {return new Dimension (max(a.width, b.width, c.width),
                        a.height+b.height+c.height);}

 public static Dimension sizeDims (Dimension a, Dimension b, Dimension c,
                                   Dimension d)
 {return new Dimension (max(a.width, b.width, c.width, d.width),
                        a.height+b.height+c.height+d.height);}

 public static Dimension sizeDims (Dimension a, Dimension b, Dimension c,
                                   Dimension d, Dimension e)
 {return new Dimension
              (max(a.width, b.width, c.width, d.width, e.width),
              a.height+b.height+c.height+d.height+e.height);}

 public static Dimension sizeDims (Dimension a, Dimension b, Dimension c,
                                   Dimension d, Dimension e, Dimension f)
 {return new Dimension
              (max(a.width, b.width, c.width, d.width, e.width, f.width),
              a.height+b.height+c.height+d.height+e.height+f.height);}
} // end Display
```

---

```
/* DisplayWindow.java
Contains DisplayWindow class and SelfSizedWin interface
DisplayWindow is the superclass for all windows.
SelfSizedWin is an interface used by classes that automatically size
themselves after they have been placed on the desktop.
*/

package WinPack;
import java.awt.*;
import java.util.StringTokenizer;
import StringCode;

interface DisplayWindow extends ImageLoader {
  public final static boolean DEBUG=true;   //DK 7/12
  public final static StringCode sc = Desktop.sc;

  public abstract boolean isEmbedded ();

  public abstract String Type();
}

/*****************************************************************/
interface SelfSizedWin
//for windows that must size themselves when laid onto the desktop
{
  public Dimension size();
}
```

---

```
/*EmbeddedCenteredWindow.java
contains only EmbeddedCenteredWindow class

This is a subclass of EmbeddedWindow, which is a subclass of display window.
This class centers all text in the window.
*/

package WinPack;
import java.awt.*;
import java.util.StringTokenizer;
import StringCode;

/*****************************************************************************/
//SINCE THIS TYPE GETS TESTED FOR A LOT, MAKE IT THE MOST ABSTRACT CLASS.
//AND PUT THE BEHAVIOR IN THE WINDOWS WHICH DO THE REAL WORK.

abstract class EmbeddedCenteredWindow extends EmbeddedWindow
{
// FIELDS
  //protected CommentTitleBar titlebar;
  protected TitleBar titlebar;
//  protected ButtonBar buttonbar = new ButtonBar();

//CONSTRUCTORS
  protected EmbeddedCenteredWindow (String title)
  //THINK THIS NEEDS TO HAVE NO ARGS
  //protected EmbeddedWindow ()
  //sets up layout
  {

//    setLayout (new BorderLayout());
//   setBackground(Color.black);
//   setForeground(Color.white);
   titlebar = new TitleBar (title);

   add ("North",titlebar);
//   buttonbar.setLayout (new FlowLayout());
//   buttonbar.setBackground(Color.black);
//   buttonbar.setForeground(Color.white);
//   add("South", buttonbar);

  }  //EmbeddedCenteredWindow

 protected EmbeddedCenteredWindow (String title, Color bgcolor)
 {
   this(title);
   titlebar.setBackground(bgcolor);
 }

//FUNCTIONS
 public Image getImage(String image_url_st) {
   return ((ImageLoader) getParent()).getImage(image_url_st);
 }

 public Image getImage(java.net.URL image_url) {
   return ((ImageLoader) getParent()).getImage(image_url);
 }

 public boolean matches (String wintype,String params)
  //used so that windows can be placed on Desktop according to given layout
 {return wintype.equals(Type()) && params.equals(identifiers());}

 public boolean isEmbedded () {return true;}

 public abstract String identifiers();

 public abstract boolean MulWinsAllowed();  //indicates whether win defaults
          //in Desktop.ClosedWinLayout (and in the .layout file)
          //apply to multiple windows (of same type), or just one window
                                         //windows of given type are allowed
}  //EmbeddedCenteredWindow
```

---

```
/*EmbeddedLeftJustifiedWindow.java
contains only EmbeddedLeftJustifiedWindow class

This is a subclass of EmbeddedWindow, which is a subclass of display window.
This class justifies all text to the left side of the window.
*/

package WinPack;
```

```
import java.awt.*;
import java.util.StringTokenizer;
import StringCode;

/****************************************************************************/
//SINCE THIS TYPE GETS TESTED FOR A LOT, MAKE IT THE MOST ABSTRACT CLASS.
//AND PUT THE BEHAVIOR IN THE WINDOWS WHICH DO THE REAL WORK.

abstract class EmbeddedLeftJustifiedWindow extends EmbeddedWindow
{
// FIELDS
  protected CommentTitleBar titlebar;
//  protected ButtonBar buttonbar = new ButtonBar();

//CONSTRUCTORS
  protected EmbeddedLeftJustifiedWindow (String title)
  {
//    setLayout (new BorderLayout());
//    setBackground(Color.black);
//    setForeground(Color.white);
    titlebar = new CommentTitleBar (title);
    add ("North",titlebar);
//    buttonbar.setLayout (new FlowLayout());
//    buttonbar.setBackground(Color.black);
//    buttonbar.setForeground(Color.white);
//    add("South", buttonbar);
  }  //EmbeddedLeftJustifiedWindow

//FUNCTIONS
  public Image getImage(String image_url_st) {
    return ((ImageLoader) getParent()).getImage(image_url_st);
  }

  public Image getImage(java.net.URL image_url) {
    return ((ImageLoader) getParent()).getImage(image_url);
  }

  public boolean matches (String wintype,String params)
   //used so that windows can be placed on Desktop according to given layout
   {return wintype.equals(Type()) && params.equals(identifiers());}

  public boolean isEmbedded () {return true;}

  public abstract String identifiers();

  public abstract boolean MulWinsAllowed(); //indicates whether win defaults
          //in Desktop.ClosedWinLayout (and in the .layout file)
          //apply to multiple windows (of same type), or just one window
                                          //windows of given type are allowed
} //EmbeddedLeftJustifiedWindow
```

---

```
/* EmbeddedWindow.java
contains only EmbeddedWindow class.

This class is a subclass of display window.  It is the abstract class for
any window that is embedded in Desktop (see Desktop.java).  It was created
for GTT to distinguish it from ForegroundWindows, which are displayed as
a separate window, outside the browser.  ForegroundWindows are not used in
DDS.
*/

package WinPack;
import java.awt.*;
import java.util.StringTokenizer;
import StringCode;

/****************************************************************************/
abstract class EmbeddedWindow extends Panel implements DisplayWindow
{
// FIELDS
  //protected CommentTitleBar titlebar;
  //protected TitleBar titlebar;
  protected ButtonBar buttonbar = new ButtonBar();

//CONSTRUCTORS
  //protected EmbeddedWindow (String title)
  //THINK THIS NEEDS TO HAVE NO ARGS
  protected EmbeddedWindow ()
  //sets up layout
   {
    setLayout (new BorderLayout());
    setBackground(Color.black);
    setForeground(Color.white);
//   titlebar = new CommentTitleBar (title);
//   add ("North",titlebar);
    buttonbar.setLayout (new FlowLayout());
//   buttonbar.setBackground(Color.black);
//   buttonbar.setForeground(Color.black);
    add("South", buttonbar);
   } //EmbeddedWindow

//FUNCTIONS
  public Image getImage(String image_url_st) {
    return ((ImageLoader) getParent()).getImage(image_url_st);
   }

  public Image getImage(java.net.URL image_url) {
    return ((ImageLoader) getParent()).getImage(image_url);
   }
```

```
 public boolean matches (String wintype,String params)
  //used so that windows can be placed on Desktop according to given layout
  {return wintype.equals(Type()) && params.equals(identifiers());}

 public boolean isEmbedded () {return true;}

 public abstract String identifiers();

 public abstract boolean MulWinsAllowed(); //indicates whether win defaults
         //in Desktop.ClosedWinLayout (and in the .layout file)
         //apply to multiple windows (of same type), or just one window
                                     //windows of given type are allowed
} //EmbeddedWindow
```

---

```
/*ErrorWindow.java
Contains ErrorWindow class.

ErrorWindows are used to display errors.  They are embedded in the Desktop
(see Desktop.java).  They include a message and "OK" button and close when
the OK button is pressed.
*/

package WinPack;
import java.awt.*;
import PromptFrame;

class ErrorWindow extends EmbeddedCenteredWindow implements SelfSizedWin
//This window is used to display errors
/*will be changed to ForegroundWindow when that class works*/
{
// FIELDS
  protected Button OKButton = new Button("OK");
  LabelSet message;
  boolean finalWin; //true if the PromptFrame should be destroyed
         //after user hits OK button

// CONSTRUCTORS
  public ErrorWindow(String title, LabelSet msg)
  {
    super (title);
    init (msg, false);
/* NOW DONE BY init()
    if (DEBUG) System.out.println("ErrorWindow:init");

    add("Center", (message=msg));
    OKButton = new Button("OK");
    buttonbar.add(OKButton);
*/
  } // end ErrorWindow constructor
```

```
 public ErrorWindow (String title, LabelSet msg, boolean fw)
 {
  super(title);
  init(msg,fw);
 }

//METHODS
 private void init (LabelSet msg, boolean fw)
 {
  finalWin = fw;
  add("Center", (message=msg));
  OKButton = new Button("OK");
  buttonbar.add(OKButton);
 }

 public boolean handleEvent(Event evt)
 {
  if (evt.id == Event.ACTION_EVENT) {
   if (evt.target.equals(OKButton))
    if (finalWin)
     ((PromptFrame)getParent().  //Desktop
       getParent().        //ConfFrame
       getParent()).        //PromptFrame
       destroy();
    else
     ((Desktop)getParent()).removeWindow(this);
  }   //if evt.id
  return super.handleEvent(evt);
 } // end handleEvent

 public String Type() {return sc.ERRWINSTRING;}

 public String identifiers() {return " ";}

 public boolean MulWinsAllowed() {return true;}

 public Dimension size() {
  Dimension s = Display.sizeDims(titlebar.stringSize(),
                              message.size(), buttonbar.size());
        s.height += 30;    //give it some padding
        s.width += 30;
        return s;
 } //size()
} // end ErrorWindow
```

---

```
/*
** GraphicFont class 1.0
** 2/22/96 Kevin Hughes, kevinh@eit.com
** Copyright (c) 1996 VeriFone, Inc.
```

```
**
** Modified from the Sun license:
**
** Permission to use, copy, modify, and distribute this software
** and its documentation for COMMERCIAL and NON_COMMERCIAL purposes
** and without fee is hereby granted provided that this copyright notice
** appears in all copies.
**
** VERIFONE MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
OF
** THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
** TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
** PARTICULAR PURPOSE, OR NON-INFRINGEMENT. VERIFONE SHALL NOT BE LIABLE FOR
** ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
** DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
**
** Should you make any improvements, please contact the original author.
**
** Limitations:
**
**   * There is no way to retrieve the name, size, or style of these
**     fonts. It would be nice if this information could be put in the
**     comment area of the image, (as well as pointers to T1/other versions)
**     but I have no Java code yet to extract GIF comments.
**   * The font format has no magic number; GF crashes if given a bad image.
**   * Kerning is not supported.
**   * Other non-GF font encodings (PC, Mac, Java) have not been fully tested.
**   * Fonts cannot be drawn in multiple colors.
**   * Up/down and right-to-left rendering is not supported.
**   * You cannot combine glyphs to make others.
**   * This file probably has more lines of comments than code. :)
*/

import java.awt.*;
import java.awt.image.*;
import java.util.*;

/**
 * A class that allows the use of a font encoded in the GraphicFont format.
 * It allows much of the same functionality as Sun's Font and FontMetrics
 * classes.
 *
 * For example:
 * <pre>
 *   g = getGraphics();
 *   GraphicFont gf = new GraphicFont(fontImage);
 *   gf.setFgColor(Color.green);
 *   gf.drawString(g, "This is a string drawn with GraphicFont.", 10, 10);
 *   int ascent = gf.getAscent();
 * </pre>
 *
 * @see         awt.Font
 * @see         awt.FontMetrics
 * @version     1.0 February 22, 1996
 * @author      Kevin Hughes, kevinh@eit.com
 */

public class GraphicFont
{
        private int DEFAULT_TAB_WIDTH = 5;
        private int UNKNOWN_CHAR = 42; // an asterix
        private int NO_CHAR = -1;
        private int BLANK_PIXEL = 0x00000000;
        private int OPAQUE_PIXEL = 0xff000000;
        private int BLACK_PIXEL = 0xff000000;
        private int WHITE_PIXEL = 0x00ffffff;
//      private float CONTRAST = (float) 0.2;
        private float CONTRAST = (float) 0.0;

// The CONTRAST value increases the foreground/background contrast by a fixed
// percent, in order to see antialiased text more clearly against colored
// backgrounds.

        private int GF_ORDER = 0;
        private int JAVA_SHORT_ORDER = 1;
        private int JAVA_LONG_ORDER = 2;
        private int MAC_ORDER = 3;
        private int PC_ORDER = 4;
        private int CUSTOM_ORDER = 5;

        private Image fontImage;
        private int imageByteArray[];
        private int imageWidthArray[];
        private int imageByteOffsetArray[];

        private int fontChars;
        private int fontOrder;

        private int totalHeight;
        private int fontHeight;
        private int ascent;
        private int descent;
        private int wordSpace;
        private int charSpace;
        private int leading;
        private int maxAscent;
        private int maxDescent;
        private int tabWidth;

// These font tables describe how the fonts are written in the
// encoded format, with the codes equal to Java font characters.

// The character sequence for the basic GraphicFont encoding format.
// This helps translate the characters into the proper Java character
// numbering scheme.

        private int gfCharMap[] = {
                'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
                'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
                'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
                'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
                'w', 'x', 'y', 'z', '0', '1', '2', '3', '4', '5', '6', '7',
                '8', '9', '~', '!', '@', '#', '$', '%', '^', '&', '*', '(',
                ')', '_', '+', '|', '{', '}', ':', '"', '<', '>', '?', '`',
                '-', '=', '\\', '[', ']', ';', '\'', ',', '.', '/'
                };

// Side note: Java fonts are apparently *not* ISO-Latin-1 (ISO8859-1).
// The font set is off by just a few characters.

// Damn those Mac fonts; they're all screwed up.
// This table starts at ASCII value 127. Unknown characters with no Java
// equivalent are replaced with asterisks (ASCII value 42).

        private int macCharMap[] = {
                42, 196, 197, 199, 201, 209, 214, 220, 225, 224, 226, 228,
                227, 229, 231, 233, 232, 234, 235, 237, 236, 238, 239, 241,
                243, 242, 244, 246, 245, 250, 249, 251, 252, 134, 176, 162,
                163, 167, 149, 182, 223, 174, 169, 153, 180, 168, 42, 198,
                216, 42, 177, 42, 42, 165, 181, 42, 42, 42, 42, 42, 170, 186,
                42, 230, 248, 191, 161, 172, 42, 131, 42, 42, 171, 187, 133,
                42, 192, 195, 213, 140, 156, 150, 151, 147, 148, 145, 146,
                247, 42, 255, 159, 42, 164, 139, 155, 42, 42, 135, 183, 42,
                132, 137, 194, 202, 193, 203, 200, 205, 206, 207, 204, 211,
                212, 42, 210, 218, 219, 217, 42, 136, 152, 175, 42, 42, 42,
                184, 42, 42, 42
                };

// If a character cannot be found, this table is used to try to make
// a reasonable substitution from the standard ASCII set.

        private int substitutionMap[] = {
                131, 102, 138, 83, 139, 60, 145, 39, 146, 39, 147, 34,
                148, 34, 150, 45, 151, 45, 152, 126, 154, 115, 155, 62,
                159, 89, 166, 124, 169, 99, 173, 45, 174, 114, 181, 117,
                192, 65, 193, 65, 194, 65, 195, 65, 196, 65, 197, 65,
                199, 67, 200, 69, 201, 69, 202, 69, 203, 69, 204, 73,
                205, 73, 206, 73, 207, 73, 209, 78, 210, 79, 211, 79,
                212, 79, 213, 79, 214, 79, 215, 120, 217, 85, 218, 85,
                219, 85, 220, 85, 221, 89, 223, 66, 224, 97, 225, 97,
                226, 97, 227, 97, 228, 97, 229, 97, 231, 99, 232, 101,
                233, 101, 234, 101, 235, 101, 236, 105, 237, 105,
                238, 105, 239, 105, 240, 111, 241, 110, 242, 111,
                243, 111, 244, 111, 245, 111, 246, 111, 247, 47, 249, 117,
                250, 117, 251, 117, 252, 117, 253, 121, 255, 121, -1, -1
                };

        private Hashtable charMap;

        private int fgColorRed;
        private int fgColorGreen;
        private int fgColorBlue;
        private int bgColorRed;
        private int bgColorGreen;
        private int bgColorBlue;

        private boolean hasBackground;

/**
 * Initializes a new GraphicFont with the encoded font passed to it.
 * Font metrics information is initialized with the default values as
 * suggested by the font's author.
 * The font image must be completely loaded beforehand.
 * @param loadedImage   An image encoded in the GraphicFont format
 */

        public GraphicFont(Image loadedImage)
        {
                setFont(loadedImage);
        }

/**
 * Sets a new font. Font metrics information is set to the default
 * values as suggested by the font's author.
 * The font image must be completely loaded beforehand.
 * @param loadedImage   An image encoded in the GraphicFont format
 */

        public void setFont(Image loadedImage)
        {
                int i, j;

// First, the font values and arrays are initialized.
// The foreground is black, the background is white but is not set.

                fontImage = loadedImage;
                fgColorRed = 0;
                fgColorGreen = 0;
                fgColorBlue = 0;
                bgColorRed = 255;
```

```
                    bgColorGreen = 255;
                    bgColorBlue = 255;
                    hasBackground = false;

// The font image is converted into a byte array.

                    int fontImageWidth = fontImage.getWidth(null);
                    int fontImageHeight = fontImage.getHeight(null);
                    fontHeight = fontImageHeight - 2;
                    imageByteArray = new int[fontImageWidth * fontImageHeight];
                    PixelGrabber pg = new PixelGrabber(fontImage, 0, 0,
                         fontImageWidth, fontImageHeight, imageByteArray,
                         0, fontImageWidth);

                    try {
                         pg.grabPixels();
                         while ((pg.status() & ImageObserver.ALLBITS) == 0)
                                   ;
                    } catch (InterruptedException e) {
                         ;
                    }

// The suggested default font metrics are loaded.

                    totalHeight = imageByteArray[0] >> 16 & 0xff;
                    ascent = imageByteArray[0] >> 8 & 0xff;
                    descent = imageByteArray[0] >> 0 & 0xff;
                    wordSpace = imageByteArray[1] >> 16 & 0xff;
                    charSpace = imageByteArray[1] >> 8 & 0xff;
                    leading = imageByteArray[1] >> 0 & 0xff;
                    maxAscent = imageByteArray[2] >> 16 & 0xff;
                    maxDescent = imageByteArray[2] >> 8 & 0xff;
                    fontOrder = imageByteArray[2] >> 0 & 0xff;
                    tabWidth = (wordSpace + charSpace) * DEFAULT_TAB_WIDTH;

// How many characters are described in this font?

                    fontChars = 0;
                    for (i = 0; i < fontImageWidth; i++) {
                         if (imageByteArray[i] == BLACK_PIXEL)
                              fontChars++;
                    }
                    fontChars++;

// The character separators are recognized and the widths are recorded.

                    imageWidthArray = new int[fontChars];
                    imageByteOffsetArray = new int[fontChars];
                    imageByteOffsetArray[0] = 1;
                    int width = 0;
                    for (i = 0, j = 0; i < fontImageWidth; i++) {
                              if (imageByteArray[i] == BLACK_PIXEL) {
                                        imageByteOffsetArray[j + 1] = i + 2;
                                        imageWidthArray[j++] = width;
                                        width = 0;
                              }
                              else
                                        width++;
                    }

                    imageWidthArray[j] = width;

// Character mapping is performed according to the particular order.

                    charMap = new Hashtable();
                    if (fontOrder == GF_ORDER) {
                              for (i = 0; i < 94; i++)
                                        addChar(gfCharMap[i], i);
                    }
                    else if (fontOrder == JAVA_SHORT_ORDER) {
                              for (i = 33, j = 0; i < 127; i++)
                                        addChar(i, j++);
                    }
                    else if (fontOrder == JAVA_LONG_ORDER) {
                              for (i = 33, j = 0; i < 256; i++)
                                        addChar(i, j++);
                    }
                    else if (fontOrder == MAC_ORDER) {
                              for (i = 33, j = 0; i < 127; i++)
                                        addChar(i, j++);
                              for (i = j, j = 0; i < 224; i++)
                                        addChar(macCharMap[j++], i);
                    }
                    else if (fontOrder == PC_ORDER) {
                              for (i = 0; i < 256; i++)
                                        addChar(i, i);
                    }

// Custom character mapping is then found and loaded.

                    int testPixel;
                    int index = (fontImageHeight - 1) * fontImageWidth;
                    for (i = 0; i < fontChars; i++) {
                              testPixel = imageByteArray[index +
                                        imageByteOffsetArray[i] - 1];
                              if (testPixel != WHITE_PIXEL) {
                                        int value = 0x00ffffff & testPixel;
                                        addChar(value, i);
                              }
```

```
		}
	}

/**
 * Gets the total height of the current font, which amounts to the
 * ascent plus the descent plus the leading.
 * @return		The total height of the font in pixels
 */

	public int getHeight()
	{
		return totalHeight;
	}

/**
 * Gets the ascent of the current font, the distance from the baseline
 * to the top of the regular characters (typically A-Z, a-z, 0-9).
 * @return		The typical ascent in pixels
 */

	public int getAscent()
	{
		return ascent;
	}

/**
 * Gets the descent of the current font, the distance from the baseline
 * to the bottom of the regular characters (typically A-Z, a-z, 0-9).
 * @return		The typical descent in pixels
 */

	public int getDescent()
	{
		return descent;
	}

/**
 * Gets the maximum ascent of the current font. No character in the font
 * extends above the baseline further than this.
 * @return		The maximum ascent in pixels
 */

	public int getMaxAscent()
	{
		return maxAscent;
	}

/**
 * Gets the maximum descent of the current font. No character in the font
 * descends below the baseline further than this.
 * @return		The maximum descent in pixels
 */

	public int getMaxDescent()
	{
		return maxDescent;
	}

/**
 * Gets the leading of the current font, the space between the
 * descent of one line and the ascent of the line below it.
 * @return		The leading in pixels
 */

	public int getLeading()
	{
		return leading;
	}

/**
 * Gets the word spacing of the current font. The space character
 * is remapped to this.
 * @return		The typical space between words in pixels
 */

	public int getWordSpace()
	{
		return wordSpace;
	}

/**
 * Gets the space between non-space characters.
 * @return		The typical space between characters in pixels
 */

	public int getCharSpace()
	{
		return charSpace;
	}

/**
 * Gets the tab width in how many space characters make up a tab
 * character. The default is five - five spaces per tab.
 * @return		The tab width
 */

	public int getTabWidth()
	{
		return tabWidth;
	}
```

```
/**
 * Sets the leading in pixels. The setting will only be visible when creating
 * text blocks. The leading can be set to a negative value.
 * @param i                    The leading to set the font to
 */

        public void setLeading(int i)
        {
                leading = i;
        }

/**
 * Sets the word spacing in pixels. The space character is
 * remapped to this. Negative values can be used.
 * @param i                    The amount of word spacing used in drawing strings.
 */

        public void setWordSpace(int i)
        {
                wordSpace = i;
        }

/**
 * Sets the character spacing in pixels. Negative values can be used.
 * @param i                    The amount of space between individual characters.
 */

        public void setCharSpace(int i)
        {
                charSpace = i;
        }

/**
 * Sets the tab width as the number of spaces that make up a tab character.
 * @param i                    The number of spaces that comprise a tab character
 */

        public void setTabWidth(int i)
        {
                tabWidth = i;
        }

/**
 * Sets the foreground color, the color the font characters are drawn in.
 * @param c                    The color characters will be drawn in
 */

        public void setColor(Color c)
        {
                if (c == null) {
                        fgColorRed = 0;
                        fgColorGreen = 0;
                        fgColorBlue = 0;
                }
                else {
                        fgColorRed = c.getRed();
                        fgColorGreen = c.getGreen();
                        fgColorBlue = c.getBlue();
                }
        }

/**
 * Sets the foreground color. Same as setColor(Color).
 * @param c                 The color characters will be drawn in
 */

        public void setFgColor(Color c)
        {
                setColor(c);
        }

/**
 * Sets the background color the font characters are drawn against.
 * This is very useful for displaying different backgrounds for
 * antialiased fonts.
 * @param c                 The color characters will be drawn against.
 *                          If the background is set to null, the
 *                          background will be transparent.
 */

        public void setBgColor(Color c)
        {
                if (c == null) {
                        hasBackground = false;
                        bgColorRed = 255;
                        bgColorGreen = 255;
                        bgColorBlue = 255;
                }
                else {
                        hasBackground = true;
                        bgColorRed = c.getRed();
                        bgColorGreen = c.getGreen();
                        bgColorBlue = c.getBlue();
                }
        }

/**
 * Returns the width in pixels of a given string. If a character is
 * unsupported in the font, the width of an asterisk character or
 * suitable substitue is used.
 * @param s      The desired string
 * @return       The width of the string after rendering
 */

        public int stringWidth(String s)
        {
                int length = s.length();
                int totalLength = 0;
                int i, c, width;

                for (i = 0; i < length; i++) {
                        c = s.charAt(i);
                        if (c == ' ')
                                totalLength += wordSpace;
                        else if (c == '\t')
                                totalLength += tabWidth;
                        else {
                                width = charWidth(s.charAt(i));
                                totalLength += width;
                                if (i != length - 1)
                                        totalLength += charSpace;
                        }
                }

                return totalLength;
        }

/**
 * Returns the width in pixels of a given character. If the character is
 * unknown, the width of an asterix character or suitable substitue is used.
 * @param c      The desired character
 * @return       The width of the character after rendering
 */

        public int charWidth(int c)
        {
                if (c == ' ')
                        return wordSpace;
                else if (c == '\t')
                        return tabWidth;

                return imageWidthArray[lookupChar(c)] - 2;
        }

/**
 * Returns a rendered image given a string using the current font.
 * If a character is unknown, an asterisk character is substituted.
 * Non-US characters that are not supported in the font may be substituted
 * for simpler ASCII characters.
* @param s       The desired string
* @return        The final rendered image. If no background is set,
*                the background will be transparent.
*/

        public Image stringImage(String s)
        {
                int c, i, x1, x2, width;
                int length = s.length();
                int byteArray[];

                width = stringWidth(s);
                int maxIndex = width * fontHeight;
                byteArray = new int[maxIndex];
          for (i = 0; i < maxIndex; i++)
               if (hasBackground == true)
                   byteArray[i] = OPAQUE_PIXEL | bgColorRed << 16 |
                    bgColorGreen << 8 | bgColorBlue << 0;
               else
                   byteArray[i] = BLANK_PIXEL;

                x2 = 0;
                for (i = 0; i < length; i++) {
                        c = s.charAt(i);
                        if (c == ' ')
                                x2 += wordSpace;
                        else if (c == '\t')
                                x2 += tabWidth;
                        else {
                                x1 = imageByteOffsetArray[lookupChar(c)];
                                copyCharImage(imageByteArray,
                                        fontImage.getWidth(null), x1,
                                        byteArray, width, x2,
                                        charWidth(c), fontHeight, 0);
                                if (i != length - 1)
                                        x2 += charWidth(c) + charSpace;
                        }
                }

                return Toolkit.getDefaultToolkit().createImage(new
                        MemoryImageSource(width, fontHeight, byteArray,
                        0, width));
        }

// Copies the correct characters from the font image in order to
// render text. All the nifty color manipulation and such is done here.

        private void copyCharImage(int sourceArray[], int sourceWidth,
                int sourceX, int destArray[], int destWidth, int destX,
                int cWidth, int cHeight, int startY)
```

```
{
        int x1, x2, y, sourceIndex, destIndex;
        int alpha, red, green, blue;
        boolean doCopy;

        for (y = startY; y < startY + cHeight; y++) {
                for (x1 = sourceX, x2 = destX; x1 < sourceX + cWidth;
                x1++, x2++) {
                        doCopy = true;
                        sourceIndex = ((y - startY + 1) *
                                sourceWidth) + x1;

                        alpha = sourceArray[sourceIndex] >> 24 & 0xff;
                        red = sourceArray[sourceIndex] >> 16 & 0xff;
                        green = sourceArray[sourceIndex] >> 8 & 0xff;
                        blue = sourceArray[sourceIndex] >> 0 & 0xff;

                        if (red == 0 && green == 0 && blue == 0) {
                                red = fgColorRed;
                                green = fgColorGreen;
                                blue = fgColorBlue;
                        }
                        else if (hasBackground == true) {
                                alpha = 255;
                                if (red == 255 && green == 255 &&
                                blue == 255)
                                        doCopy = false;
                                else {
                                        float percent = (float) red /
                                                (float) 255;
                                        float ipercent = ((float) 1.0 -
                                                percent + CONTRAST);
                                        if (ipercent > 1.0)
                                                ipercent = (float) 1.0;
                                        red = (int) ((percent * (float)
                                                bgColorRed) + (ipercent
                                                * (float) fgColorRed));
                                        green = (int) ((percent *
                                                (float) bgColorGreen) +
                                                (ipercent * (float)
                                                fgColorGreen));
                                        blue = (int) ((percent *
                                                (float) bgColorBlue) +
                                                (ipercent * (float)
                                                fgColorBlue));
                                }
                        }

                        if (doCopy) {
                                destIndex = (y * destWidth) + x2;
```

```
                                        destArray[destIndex] = alpha << 24 |
                                        red << 16 | green << 8 | blue << 0;
                                }
                        }
                }
        }

/**
 * Returns a rendered image of a left-justified text block given a string
 * and a width in pixels. Only spaces are wrapped. Newline characters (\n)
 * create hard breaks.
 * @param s             The desired string
 * @param width         The maximum width of the text block
 * @return              The final rendered image. If no background is set,
 *                      the background will be transparent.
 */

        public Image stringImage(String s, int width)
        {
                int c, i, x1, x2, y, cLen = 0;
                int currentWidth = 0, maxWidth = 0, lines = 1;
                int lastSpace = 0;
                int byteArray[];
                char charArray[];

                int length = s.length();
                charArray = new char [length];
                charArray = s.toCharArray();
                for (i = 0; i < length; i++) {
                        c = (int) charArray[i];
                        if (c == '\n')
                                currentWidth = 0;
                        else if (c == ' ' || c == '\t')
                                cLen = charWidth(c);
                        else
                                cLen = charWidth(c) + charSpace;

                        if (currentWidth + cLen > width) {
                                if (c == ' ')
                                        charArray[i] = '\n';
                                else
                                        charArray[lastSpace] = '\n';
                                currentWidth = 0;
                                i = lastSpace;
                        }
                        else {
                                if (c == ' ')
                                        lastSpace = i;
                                currentWidth += cLen;
                        }
                }

// This loop goes through the string again to calculate the
// correct maximum image width.

                currentWidth = 0;
                for (i = 0; i < length; i++) {
                        c = (int) charArray[i];
                        if (c == '\n') {
                                currentWidth -= charSpace;
                                if (currentWidth > maxWidth)
                                        maxWidth = currentWidth;
                                currentWidth = 0;
                                lines++;
                        }
                        else if (c == ' ' || c == '\t')
                                currentWidth += charWidth(c);
                        else
                                currentWidth += (charWidth(c) + charSpace);
                }
                c = (int) charArray[i - 1];
                if (c != ' ' && c != '\t')
                        currentWidth -= charSpace;
                if (currentWidth > maxWidth)
                        maxWidth = currentWidth;

                int height = ((fontHeight + leading) * lines) - leading;
                int maxIndex = maxWidth * height;
                byteArray = new int[maxIndex];
         for (i = 0; i < maxIndex; i++)
              if (hasBackground == true)
                  byteArray[i] = OPAQUE_PIXEL | bgColorRed << 16 |
                  bgColorGreen << 8 | bgColorBlue << 0;
              else
                  byteArray[i] = BLANK_PIXEL;

                x2 = 0;
                y = 0;
                for (i = 0; i < length; i++) {
                        c = (int) charArray[i];
                        if (c == ' ')
                                x2 += wordSpace;
                        else if (c == '\t')
                                x2 += tabWidth;
                        else if (c == '\n') {
                                x2 = 0;
                                y += fontHeight + leading;
                        }
                        else {
                                x1 = imageByteOffsetArray[lookupChar(c)];
```

```
					copyCharImage(imageByteArray,
						fontImage.getWidth(null), x1,
						byteArray, maxWidth, x2,
						charWidth(c), fontHeight, y);
					if (i != length - 1)
						x2 += charWidth(c) + charSpace;
				}
			}

			return Toolkit.getDefaultToolkit().createImage(new
				MemoryImageSource(maxWidth, height, byteArray,
				0, maxWidth));
		}

	/**
	 * Draws a rendered image of a string given a graphics context, a string,
	 * and the coordinates to draw at. Similar to Font.drawString(), the
	 * Y coordinate starts at the baseline of the font.
	 * @param g			The graphic context to draw into
	 * @param s			The desired string to render
	 * @param x			The X coordinate
	 * @param y			The Y coordinate
	 */

		public void drawString(Graphics g, String s, int x, int y)
		{
			Image stringImage = stringImage(s);
			g.drawImage(stringImage, x, y - ascent, null);
		}

	/**
	 * Draws a rendered image of a left-justified text block given a graphics
	 * context, a string, the coordinates to draw at, and a width in pixels.
	 * The Y coordinate starts at the baseline of the first line.
	 * @param g			The graphic context to draw into
	 * @param s			The desired string to render
	 * @param x			The X coordinate
	 * @param y			The Y coordinate
	 * @param width		The maximum width of the text block
	 */

		public void drawString(Graphics g, String s, int x, int y, int width)
		{
			Image stringImage = stringImage(s, width);
			g.drawImage(stringImage, x, y - ascent, null);
		}

	/**
	 * Returns a rendered image given a character.
	 * If the character is unknown, an asterix or appropriate substitute
 * is returned.
 * @param c                  The desired character
 * @return                   The rendered image of the character.
 *                           If no background is set, the background will
 *                           be transparent.
 */

        public Image charImage(int c)
        {
                return stringImage(String.valueOf(c));
        }

// Character map hash table functions.
// The key: The integer code that represents a glyph
// The value: The index at which the glyph is stored in the image

// If an index cannot be found at first, a substitution is attempted.
// If that fails, the index to the default unknown character is returned.

        private void addChar(int key, int value)
        {
                charMap.put(new Integer(key), new Integer(value));
        }

        private int getChar(int key)
        {
                Integer result;
                result = (Integer) charMap.get(new Integer(key));
                if (result == null)
                        return NO_CHAR;
                else
                        return result.intValue();
        }

        private int lookupChar(int c)
        {
                int i, value, newValue;

                value = getChar(c);
                if (value != NO_CHAR)
                        return value;
                for (i = 0; substitutionMap[i] != -1; i += 2) {
                        if (substitutionMap[i] == c) {
                                newValue = getChar(substitutionMap[i + 1]);
                                if (newValue == NO_CHAR)
                                        return getChar(UNKNOWN_CHAR);
                                else
                                        return newValue;
                        }
                }
                return getChar(UNKNOWN_CHAR);
        }
}
```

```
/*GraphicFontTextArea.java
contains GraphicFontTextArea class.

This class is jsut like a TextArea, except that it uses GraphicFont (see
GraphicFont.java).
*/

package WinPack;
import java.awt.*;
import java.util.*;
import java.net.*;
import java.applet.*;
import StreamManager;
import GraphicFont;

class GraphicFontTextArea extends Canvas implements ImageLoader {

// FIELDS
  protected String picurlstr;
  protected Vector message;
  private Image pic;  //DK 8/2
  private Image graphicTextBlock;
  //private Image fontImage = getImage(getParent().getParent().getCodeBase(), "gill.sans.12.gif");
  private MediaTracker tracker;
  private Image fontImage;
  private GraphicFont gf;
  private Graphics g;
  private Applet applet;
  private String fontImageUrlStr;
  private String messageString;
  private int vectorSize;
  private Vector messageVector;
  private int xPos = 0;
  private int yPos = 0;
  private boolean inside = false;

  private int startX, startY, endX, endY, curX, curY, deltaX, deltaY;

  //public URL fontImageUrl;

// METHODS
  public GraphicFontTextArea(Vector message)
   //throws MalformedURLException
   {
   messageVector = message;
   //System.out.println("GOT INSIDE GRAPHICFONTTEXTAREA, message is " + message);
```

```
 messageString = (String)message.elementAt(0);
 //System.out.println("Message IS: " + messageString);
 //System.out.println("MESSAGE SIZE IS " + message.size());
 vectorSize = message.size();
 //fontImageUrlStr = "file:/F:/Cafe/Projects/GTTFinalNP/aa.garamond.italic.14.gif";
 //fontImageUrlStr = "file:/H:/Cafe/Projects/GTTFinalNS2/gill.sans.12.gif";
 //fontImageUrlStr = "/mas/vlw/data/web/docs/civiscape/GTTFinalNS/gill.sans.12.gif";
 fontImageUrlStr = "http://campbell.media.mit.edu:8001/GTTFinalNS/gillSans12.GIF";
 tracker = new MediaTracker (this);
 //getPicture();
 //init();

} // end Picture constructor

public GraphicFontTextArea(String message)
 //throws MalformedURLException
 {

 //System.out.println("GOT INSIDE GRAPHICFONTTEXTAREA, string message is " + message);
 //fontImageUrlStr = "file:/H:/Cafe/Projects/GTTFinalNS2/gill.sans.12.gif";
 //fontImageUrlStr = "/mas/vlw/data/web/docs/civiscape/GTTFinalNS/gill.sans.12.gif";
 fontImageUrlStr = "http://campbell.media.mit.edu:8001/GTTFinalNS/gillSans12.GIF";
 tracker = new MediaTracker (this);

} // end Picture constructor



public void init() {
   tracker = new MediaTracker (this);
   //System.out.println("INSIDE INIT, just before try");

    try {
       //System.out.println("GETTING THE FONT IMAGE");
       //URL fontImageUrl = new URL("file:/F:/Cafe/Projects/GTTFinalNP/gill.sans.12.gif");
       //System.out.println("GOT THE URL");
       //fontImage = getImage(fontImageUrl);
       //System.out.println("GOT THE FONT IMAGE");
       //fontImage = getImage("file:/F:/Cafe/Projects/GTTFinalNP/gill.sans.12.gif");
       //fontImage = getImage("gill.sans.12.gif");
       //fontImage = getImage(getDocumentBase(), "aa.garamond.italic.14.gif");
       //fontImage = getImage(getDocumentBase(), "aa.garamond.italic.inverted3.14.gif");
       //fontImage = getImage(getDocumentBase(), "aa.garamond.italic.testinvert.14.gif");
       //System.out.println("after fontImage");
       tracker.addImage(fontImage, 0);
       //System.out.println("after first tracker");
       tracker.waitForID(0);
       //System.out.println("after second tracker");
       gf = new GraphicFont(fontImage);
       //System.out.println("just before the Descent request");
```

```
      int Descent = gf.getDescent();
      //System.out.println("MESSAGE: " + message);
      //System.out.println("leading is: " + Descent);

      gf.setFgColor(Color.white);
       int width = 258;

     //;;String messageString = (String)message.elementAt(0);
      //System.out.println("MESSAGE IS " + messageString);
      //graphicTextBlock = gf.stringImage(messageString, width);

            } catch (Exception e) {
            ;
            }
} //end init()

public void paint(Graphics g) {
 int i, nodesleft;
 if (pic == null) {getPicture(); init();}
 System.out.println("painting image");
 //g.drawImage(graphicTextBlock, 0, 0, this);
  //gf.setBgColor(Color.white);
  //gf.setFgColor(Color.black);

 for (i=0;
    i<vectorSize;
    i++)
    {

    //System.out.println("i = " + i);
    //System.out.println(" message at " + i + (String)messageVector.elementAt(0));
    //System.out.println(" message at " + i + (String)messageVector.elementAt(i));
    //System.out.println("deltaY = " + deltaY);

    //KEEP THE WINDOW FROM SCROLLING INAPPROPRIATELY
    if ((yPos+deltaY)>0) {yPos = 0; deltaY = 0;}
    if ((yPos+deltaY)<-(vectorSize*15-75) & vectorSize>6)
      {yPos = -(vectorSize*15-75);
      deltaY = 0;
      //System.out.println("vectorsize = " + vectorSize);
      }
    //System.out.println("vectorsize etc.." + (vectorSize*15-75));
    //System.out.println("YPOS = " + yPos);
    //System.out.println("Ypos + deltaY " + (yPos + deltaY));
    gf.drawString(g, (String)messageVector.elementAt(i), 0, 15+yPos+deltaY+i*14);
    }

} // end paint
```

```
//public String getURL () {return fontImage;}  //DK 7/16

public Dimension size() {
  if (pic==null) getPicture();
  //return new Dimension (fontImage.getWidth(applet),fontImage.getHeight(applet));
  if (vectorSize<7)
  return new Dimension (285, (vectorSize-1)*14);
  else
  return new Dimension (285, 5*14);
 } //DK 9/6

public Image getImage(String image_url_st) {
  return ((ImageLoader)getParent()).getImage(image_url_st);
 }

public Image getImage(java.net.URL image_url) {
  return ((ImageLoader)getParent()).getImage(image_url);
 }

public void getPicture() {
//loads picture -- can't be used in init because it causes
//NullPointerException when getImage is called (due to getParent()
//function)
  try {
    URL url = new URL(fontImageUrlStr);
    //System.out.println("URL:" + url);
    fontImage = getImage(url);
    //System.out.println("fontImage:" + fontImage);
    applet = (Applet)getParent().getParent().getParent().getParent();
    //System.out.println("Image:" + fontImage.toString());
    init();
  }
  catch(Exception e) {
    System.err.println("Exception:");
    e.printStackTrace();
  } //try/catch
 } //getPicture()



public boolean handleEvent(Event e)
   {

      //if(e.target instanceof GraphicFontTextArea)
      if (vectorSize>6)
      {
        if (e.id == Event.MOUSE_DOWN)
                        {
                                    // Save starting location...
```

```
                    startX = e.x;
                    startY = e.y;
                    //System.out.println("got mouse down");
                    inside = true;
                }
                else if (e.id == Event.MOUSE_DRAG & inside)
                {
                 //System.out.println("at a new point = " + e.x + " " + e.y);
                 curX = e.x;
                 curY = e.y;
                 //deltaX = startX - curX;
                 deltaX = curX - startX;
                 //deltaY = startY - curY;
                 deltaY = curY - startY;
                 //System.out.println("deltay =" + deltaY);
                 //repaint();
                 //System.out.println("totally inside drag");
                }
                if (e.id == Event.MOUSE_UP & inside)
                {
          yPos = yPos + deltaY;
          repaint();
          //System.out.println("just set ypos = " + yPos);
        }

      return true; //handled it
                }


   return super.handleEvent(e); // not our event
}
} // end GraphicFontTextArea
```

---

```
/* HistWindow.java
Contains HistWindow class.

This class is used to display histories. This includes all comments made,
objects imported, and discussions created by a particular person, in a
particular discussion, or both. All records (see Recorder.java)
are listed in chronological order.
*/

package WinPack;
import java.awt.*;
import java.util.*;
import StreamManager;
import StringCode;

class HistWindow extends EmbeddedCenteredWindow
{
```

```
// FIELDS
  String HistDisc;     //discussion for which history is being loaded
  String HistPerson;   //person for which history is being loaded
  public Display display;
  private Button closebutton = new Button("Close");  //gets rid of this window


// PUBLIC METHODS
  public HistWindow(String d, String p, String text, int width, int height)
  {
   super ((d.equals(StringCode.EMPTYFIELD)||
         p.equals(StringCode.EMPTYFIELD)) ? d+p:d+','+p);
   if (DEBUG) System.out.println ("HistWindow:init");
   HistDisc = d;
   HistPerson = p;
   display = new Display(text,width,height);
   add("Center",display);
   buttonbar.add(closebutton);
  } // end init

  public boolean handleEvent(Event evt)
  {
   if (evt.id == Event.ACTION_EVENT) {
     if (evt.target.equals(closebutton)) {
       Desktop d=(Desktop)getParent();
       d.removeWindow(this);
       return true;
     }     //if evt.target
   }  //if evt.id
   return super.handleEvent(evt);
  } // end handleEvent

  public void setText (String text)
  {
   display.setText (text);
  }      //setText

  public void appendText (String text)
  {display.insertString (text);}

  public static String unparseHistWinInfo (String disc, String spkr)
  {return disc+sc.IDDIVIDER+spkr;}

  public String identifiers () {
   return unparseHistWinInfo (HistDisc,HistPerson);
  }

  public boolean MulWinsAllowed() {return false;}

  public  String Type() {return sc.HISTWINSTRING;}
```

```
}        //HistWindow
```

```
/* ImageLoader.java
Contains ImageLoader interface.

This interface should be implemented by any class that will be
used to import images.
*/

package WinPack;
import java.awt.*;
import java.net.URL;

interface ImageLoader {

public java.awt.Image getImage(String image_url_st);
// note: image name must be relative to codebase url

public java.awt.Image getImage(URL image_url);

}
```

```
/*LabelSet.java
contains LabelSet class

This class is a set of Labels, stacked vertically.  It is used in
Windows to display multiple lines of text.
*/

package WinPack;
import java.awt.*;
import java.util.*;
import java.net.*;
import StreamManager;

/************************************************************/

class LabelSet extends Panel {
  private static final boolean DEBUG = DisplayWindow.DEBUG;

  private Vector message;  //each element represents a line
  private int height;  //of labelset
  private int longline;  //length of longest line of message, in pixels

          //LabelSet (when using size())

  public LabelSet() {
    setFont (ConfFrame.standardFont);
    message = new Vector (1,1);
  }
```

```
public LabelSet (String msg)
//This is a dual purpose constructor
//It can create LabelSets of one line
//or it can parse the message to create LabelSets of multiple
//   lines.  It knows whether it is supposed to be parsed
//   by the number of tokens parseMessage() returns
{
  setFont (ConfFrame.standardFont);
  StringTokenizer tok = StreamManager.parseMessage(msg);
  int numTokens = tok.countTokens();
  if (numTokens==1)
    setText(msg, false);
  else if (numTokens==2)
    setText(tok.nextToken(),tok.nextToken());
  //add more else if clauses later if necessary
}

public LabelSet (String msg, String msg2)
{
  setFont (ConfFrame.standardFont);
  setText(msg,msg2, false);
}


public LabelSet (String msg, String msg2, String msg3)
{
  setFont (ConfFrame.standardFont);
  setText (msg,msg2,msg3,false);
}

public LabelSet (Vector mess)
//mess is a Vector of Strings
{
  setFont (ConfFrame.standardFont);
  height = 0;
  setLayout(new GridLayout(mess.size(),1));
  for (Enumeration enum = mess.elements(); enum.hasMoreElements();)
    add(new Label((String)enum.nextElement(),Label.LEFT));
  message = mess;
}  //LabelSet (Vector)


public void setText (String msg, boolean pt)
//pt = indicater of whether labelset should be painted
{
  height = 0;
  message = new Vector(1,1);
  Label msglbl = new Label(msg,Label.LEFT);

  removeAll();
  setLayout (new FlowLayout());
  add (msglbl);
  message.addElement(msg);
  Dimension d = msglbl.size();
  if (pt) paintAll (getGraphics());
 }  //setText (String,boolean)

 public void setText (String msg, String msg2, boolean pt)
 {
  message = new Vector (2,1);
  height = 0;
  removeAll();
  setLayout(new GridLayout(2,1));
  add(new Label(msg,Label.LEFT));
  add(new Label(msg2,Label.LEFT));
  message.addElement(msg);
  message.addElement(msg2);

  if (pt) paintAll(getGraphics());
 }  //LabelSet (String,String,boolean)

 public void setText (String msg, String msg2, String msg3, boolean pt)
 {
  message = new Vector (3,1);
  height = 0;
  setLayout(new GridLayout(3,1));
  add (new Label(msg,Label.LEFT));
  add(new Label(msg2,Label.LEFT));
  add(new Label(msg3,Label.LEFT));
  message.addElement(msg);
  message.addElement(msg2);
  message.addElement(msg3);

  if (pt) paintAll (getGraphics());
 }  //LabelSet (String,String,String,boolean)

 public void setText (String msg) {setText (msg, true);}
 public void setText (String msg, String msg2) {setText(msg, msg2, true);}
 public void setText (String msg, String msg2, String msg3)
 {setText(msg, msg2, msg3, true);}

 public Dimension size()
 {
  if (height==0)  //first time determining size() for this text;
  {
   int numlines = message.size();
   FontMetrics f = getGraphics().getFontMetrics();
//    height = numlines * 30;
   height = numlines*f.getHeight()+(numlines+1)*Display.MARGIN;
   if (numlines ==0) {
```

```
            if (DEBUG) System.out.println ("Size of LabelSet():0,"+height);
            return new Dimension (0,height);
      } //if numlines
      Enumeration messenum = message.elements();
      longline = f.stringWidth((String)messenum.nextElement());
      int nextlen;
      for (;--numlines>0;)
        if ((nextlen=f.stringWidth((String)messenum.nextElement()))>longline)
              longline = nextlen;
    } //if height==0
    if (DEBUG) System.out.println ("Size of LabelSet("+
            message.firstElement()+"): "+longline+','+height);
    return new Dimension (longline, height);
  } //size()
} //LabelSet
```

```
/* PromptFrame.java
Contains PromptFrame and PromptInLoop classes

This file is embedded in PromptFrame.html, which is in the prompt frame
of DDS.html.
This class contains the control panel for DDS.  It behaves almost exactly
like GTT, except that it doesn't support CommentWindows and PictureWindows.
This control panel can be minimized so that the VRML world has more space
on the page.
When this class is loaded, it contacts the PromptServer (see
PromptServer.java) on the server side.  It also creates a BCInLoop (which is
a subclass of InputLoop -- see InputLoop.java) to wait on the messages sent
from the server.
*/

import java.net.*;
import java.io.*;
import java.awt.*;
import java.applet.*;
import java.lang.*;
import java.util.*;
import InputLoop;
import StringCode;
import WinPack.ConfFrame;

public class PromptFrame extends Applet
// This class opens up connection to the ConferenceServer,
// and provides a control panel interface (ConfFrame) for the discussion
{

// CONSTANTS
  public static final boolean DEBUG = true;
  public static final StringCode sc = new StringCode();
  private static int FRAMEWIDTH;
  private static int FRAMEHEIGHT;

// FIELDS
  public Socket socket;
  public DataInputStream input;
  public StreamManager output;
  public ConfFrame cframe;
  public PromptInLoop input_loop;

// PUBLIC METHODS
  public void init()
  // effects:  create socket and associate input and
  // output streams with it.  Then start input loops
  // to constantly check for input from the server
  // Also creates initial user interface.
  {
    super.init();
    dprint("super initialized in PromptFrame");
    FRAMEWIDTH = Integer.parseInt(getParameter("WIDTH"));
    FRAMEHEIGHT = Integer.parseInt(getParameter("HEIGHT"));
    dprint ("Desktop dimensions:"+
           FRAMEHEIGHT+','+FRAMEWIDTH);
    try {
      socket = new Socket(sc.HOSTNAME, sc.PROMPT_PORT);
      input = new DataInputStream(socket.getInputStream());
      output =
           new StreamManager(new PrintStream(socket.getOutputStream()));
      cframe = new ConfFrame(sc.DEFAULTDISC,"",output,
           FRAMEWIDTH, FRAMEHEIGHT, this, sc.HOSTNAME);
      add(cframe);

      input_loop = new PromptInLoop(input, cframe);
      input_loop.start();
      output.send(sc.NEWUSERSTRING);
    }
    catch (SocketException e) {
      dprint ("Server down; notifying user");
      cframe = new ConfFrame(sc.DEFAULTDISC,"",(StreamManager)null,
              FRAMEWIDTH, FRAMEHEIGHT, this, sc.HOSTNAME);
      add(cframe);
      cframe.addFinalWin ("SERVER UNREACHABLE",
         "  Sorry, the server can not currently be contacted");
//     stop();
    }
    catch(Exception e) {
      StreamManager.showError(e);
      try {
        socket.close();
      }
      catch(Exception e2) {StreamManager.showError(e2);}
      System.exit(1);
    }
  }  // end init

  public void destroy()
  // effects: destroys all threads that have been created by this applet
  {
    try {
      dprint ("PromptFrame: destroy()");
      remove(cframe);
      cframe.dispose();
      input_loop.stop();
      cframe.output.send (sc.DONESTRING);
      stop();
    } catch (Exception e) {StreamManager.showError(e);}
  } // end destroy

  public void dprint (String s)
  {if (DEBUG) System.out.println (s);}

} // end PromptFrame

/**************************************************************/

class PromptInLoop extends InputLoop {
//  This class constantly checks the input stream to see  if
//  the server has sent anything

// FIELDS
  public ConfFrame cframe;

// METHODS

  public PromptInLoop(DataInputStream in, ConfFrame cf)
  {
    super(in);
      cframe = cf;
  } // end PromptInLoop constructor

  public boolean process_message(String server_message)
  // effects:  if input from server is a command of some sort, (such as
  // a command to create a new window), then this method processes it.
  // To add a new command that the client will respond to, simply
  // add another "if" statement below.
  {
    if (PromptFrame.DEBUG) System.out.println("processing in promptframe:"+
                                      server_message);

  try {
    StringTokenizer tok = StreamManager.parseMessage(server_message);
```

```
String code = tok.nextToken();

//FOR SETTING UP NEW USERS
if (code.equals(sc.PERSONCAPSTRING)) {
 //the number of people using the server is at capacity
 cframe.addFinalWin ("DDS FULL",
          "Sorry, DDS is currently full.");
 return true;
}  //if code.equals PERSONCAPSTRING

if (code.equals(sc.GETNAMESTRING)) {
 //room for more users -- gets name of new user
 cframe.setDiscAndPers(tok.nextToken(),   //list of all discussions
                tok.nextToken());  //list of all people
 cframe.getName("Welcome to the Design Discussion Space!");
 return true;
}   //if code.equals GETNAMESTRING

if (code.equals(sc.NEWNAMESTRING)) {
 //name entered is in database/on disk -- let user enter password
 cframe.newPwd();
 return true;
}   //if code.equals NEWNAMESTRING

if (code.equals(sc.ACTIVENAMESTRING)) {
 //a person under given name is already in conference
 cframe.getName
       ("Sorry, someone with that name is currently in the conference.");
}  //if code.equals ACTIVENAMESTRING

if (code.equals(sc.GETPWDSTRING)) {
 //name entered is already in database -- verify password
 cframe.askPwd(tok.nextToken());
 return true;
}   //if code.equals GETPWDSTRING

if (code.equals(sc.NEWUSERSTRING)) {
 //another user has logged in for first time
 //update cframe's Person List
 cframe.addPerson (tok.nextToken()); //name of person
 return true;
}  //if code.equals NEWUSERSTRING

//WINDOWS/IMPORTED OBJECTS
if (code.equals(sc.NEWWINSTRING)) {
 //create new window
 return cframe.addWindow(StreamManager.restOf(tok));
}   //if code.equals NEWWINSTRING

if (code.equals(sc.COMMENTSTRING)) {
 //someone posted a comment
 //update history windows if necessary
 cframe.addComment (tok.nextToken(),    //discussion name
                    tok.nextToken(),        //speaker
                    tok.nextToken());       //comment text
 return true;
}      //if code.equals COMMENTSTRING

if (code.equals (sc.PICWINSTRING)) {
 //user imported an object
 //add it to list of objects
 cframe.addImage (tok.nextToken(),        //title of import
                tok.nextToken());     //url
 return true;
}      //code.equals PICWINSTRING

if (code.equals (sc.OBJSTRING)) {
 //user imported an object
 //add it to list of objects
 cframe.add3DObj (tok.nextToken(),        //title of import
                tok.nextToken());     //url
 return true;
}      //code.equals PICWINSTRING

if (code.equals (sc.REMOVECOMM)) {
 cframe.removeComment (tok.nextToken(), //discussion
                    tok.nextToken());  //id
 return true;
}      //code.equals REMOVECOMM

if (code.equals (sc.REMOVEIMPORT)) {
 //importe (comment/picture/etc.) has been removed
 cframe.removeImport (tok.nextToken(),     //object type
                   tok.nextToken());  //identifier
 return true;
}      //if code.equals REMOVEIMPORT

if (code.equals(sc.SETTEXT)) {
 //sets text in TalkWindow of given speaker
 cframe.setText (tok.nextToken(),      //speaker
            tok.nextToken(),        //discussion
            tok.nextToken());       //text
 /*update chatting history*/
 return true;
}   //if code.equals SETTEXT

if (code.equals(sc.CLOSECHAT)) {
 //closes TalkWindow of given speaker
 cframe.closeTalkWin (tok.nextToken());
 return true;
```

```
    }  //if code.equals CLOSECHAT


    //DISCUSSIONS
    if (code.equals(sc.ADDSUBDISC)) {
      //new discussion created
      //if discussion's parent is cframe's discussion, then add it to
      //   the DiscChoice menu
      cframe.addSubDisc (tok.nextToken(),       //parent of new discussion
                    tok.nextToken());       //name of new discussion
      return true;
    }   //if code.equals ADDSUBDISC

    if (code.equals(sc.ADDSUPERDISC)) {
      //create new button for parent discussion
      cframe.addSuperDisc (tok.nextToken());
      return true;
    }   //if code.equals ADDSUPERDISC

    if (code.equals(sc.CLEARDESKTOP)) {
      cframe.setInDatabase();
      cframe.reset(tok.nextToken());
      return true;
    }   //if code.equals CLEARDESKTOP

    if (code.equals(sc.DISCFULLSTRING)) {
      //Discussion is full; notify user
//    cframe.showDiscFull();
      cframe.addWindow (StreamManager.unparseMessage(sc.ERRWINSTRING,
          "NO ROOM",
          "  Sorry, there is no room in that discussion"));
      return true;
    }   //if code.equals DISCFULLSTRING

    if (code.equals(sc.MAXDISCSTRING)) {
      //conference has reached maximum number of allowable discusssions
//    cframe.showMaxDisc();
      cframe.addWindow (StreamManager.unparseMessage(sc.ERRWINSTRING,
          "NO ROOM",
          "  Sorry, there is no room to create another discussion"));
      return true;
    }  //if code.equals MAXDISCSTRING

    if (code.equals(sc.COMMENTLIST)) {
      //list of comments for the discussion the user just
      //switched into
      cframe.setComments(tok.nextToken());
    }

  } catch(Exception e) {StreamManager.showError(e);}
  System.err.println ("Message not processed: "+server_message);
  return false;

  } // end process_message

  protected void handleSocketException ()
  //called when server crashes, or client gets disconnected
  //for some reason
  {
    System.err.println
      ("Connection with host lost; notifying user");
//   cframe.showConnectionLost();
    cframe.addFinalWin ("CONNECTION LOST",
      "  The connection with the server has been lost");
//   super.handleSocketException();
  }
} // end PromptInLoop
```

---

```
/* PromptWindow.java
Contains PromptWindow, NameWindow, NewPwdWindow, AskPwdWindow, NewPicWin,
   RemovePrompt, NewDiscWin, and Prompter classes.

PromptWindow is used to prompt users for information.
NameWindow (subclass of PromptWindow) asks for the user's login name.
NewPwdWindow (subclass of PromptWindow) is created if the user's login name
   is new.  It asks for the user to choose a password.
AskPwdWindow (subclass of PromptWindow) is created if the user's login name
   matches someone who has logged in before.  It asks for the password.
NewPicWin (subclass of PromptWindow) is created when the user wants to import
   an image or 3-D object.  Its name carries over from GTT, when 3-D
   objects were not supported.
RemovePrompt (subclass of PromptWindow) is created when the user
   wants to remove a comment, image, or 3-D object from the discussion.
NewDiscWin (subclass of PromptWindow) is created when the user wants to
   create a subdiscussion.
NewHistWin (subclass of PromptWindow) is created when the user wants to
   open a new history window.
Prompter is a component of all promptwindows.  It consists of a prompt and
   a TextField (package java.awt) in which the user enters information.
*/

package WinPack;
import java.awt.*;
import java.io.PrintStream;
import java.net.URL;
import java.net.MalformedURLException;
import java.util.*;
import StreamManager;
import StringCode;
import PromptFrame;
```

```
abstract class PromptWindow extends EmbeddedCenteredWindow implements SelfSizedWin
//used to prompt users for random stuff
{
// CONSTANTS
  public static final Color BKCOLOR = Color.gray;  //background color of window
  public static final Color TEXTCOLOR = Color.black; //color of text
  public static final Color PROMPTBKCOLOR = Color.black; //colors of
  public static final Color PROMPTTEXTCOLOR = Color.white;  //prompts

// FIELDS
  protected LabelSet Text;
  protected Prompter PromptPanel;
  protected Panel CenterPanel = new Panel();
  protected TextField Input = new TextField (12);
  protected Button OKButton = new Button ("OK");
  protected Button CancelButton = new Button ("Cancel");
  protected String MessageCode;
  protected StreamManager Output;

// METHODS
  public PromptWindow(String title, LabelSet text,
                                    String promptstr, String codestr, StreamManager out)
  //textstr = text above prompt & Input textfield
  //promptstr = Prompting string
  //codestr = String code to be attached to message sent through out
  {
/*THIS LINE TO B USED WHEN FOREGROUND WINDOWS WORK
    super(parent, title, true);*/
    super(title);
    dprint("PromptWindow:init");
    MessageCode = codestr;
    Output = out;
    Text = text;

    PromptPanel= new Prompter (promptstr,Input);
    setBackground(BKCOLOR);
    setForeground(TEXTCOLOR);
//    Input.setBackground (PROMPTBKCOLOR);
//    Input.setForeground (PROMPTTEXTCOLOR);
    add ("Center", CenterPanel);
    buttonbar.add(OKButton);
    buttonbar.add(CancelButton);
//doesn't work
    Input.requestFocus();
  } // end PromptWindow constructor

  public boolean handleEvent(Event evt)
  //if OKButton is hit, sends message through Output & sets Target
  {
   if (evt.id == Event.KEY_PRESS && evt.key == 10) {
    dprint ("Return pressed");
    evt.id = Event.ACTION_EVENT;
    return handleEvent(new Event(OKButton,evt.id,evt.arg));
         //convert to an OK press
   }   //if evt.id
   if (evt.id == Event.ACTION_EVENT) {
    if (evt.target.equals(OKButton)) {
     dprint("OK pressed in PromptWindow");
     Output.send(MessageCode,Input.getText());
     ((Desktop)getParent()).removeWindow(this);
     return true;
    }  //if target.equals OKButton
    if (evt.target.equals(CancelButton)) {
     dprint ("Cancel pressed in PromptWindow");
     ((Desktop)getParent()).removeWindow(this);
     return true;
    }  //if target.equals CancelButton
   }   //if evt.id
   return super.handleEvent(evt);
  }       //handleEvent

  public String Type() {return sc.PROMPTWINSTRING;}

  public String identifiers() {return sc.EMPTYFIELD;}

  public boolean MulWinsAllowed() {return true;}

  public Dimension size()
  //used to resize window after it's placed on the desktop.
  //the formula is rather arbitrary.
  {
   dprint ("Getting size of PromptWindow");

   Dimension tsize = Text.size();
   Dimension psize = PromptPanel.size();

   return (new Dimension (Display.max (tsize.width, psize.width)+ 50,
                          Display.max (tsize.height, psize.height)*3));
  } //size()

  protected void dprint (String s)
  {if (DEBUG) System.out.println (s);}
 } // end PromptWindow

/************************************************************/

class NameWindow extends PromptWindow {
  PromptFrame Client;
  Vector plist; //vector of people in discussion; used to find out whether
```

```
      //user enters a name similar to a previous one

  public NameWindow (StreamManager out, PromptFrame client,
                    String text, Vector pl)
   {
    super ("WELCOME", new LabelSet(text), "Please enter your name:",
          sc.USERNAMESTRING, out);
    dprint ("NameWindow:init");
    Client = client;
    plist = pl;

//   setForeground(Color.black);
    CenterPanel.setLayout(new GridLayout(2,1));
    CenterPanel.add (Text);
    CenterPanel.add(PromptPanel);
    reshape(200,200,200,500);
//doesn't work
//   Input.requestFocus();
   }

  public boolean handleEvent (Event evt)
   {
    Object target = evt.target;

    if (evt.id == Event.ACTION_EVENT) {
     if (target.equals(OKButton)) {
      dprint("OK pressed in NameWindow");
      String u = Input.getText();
             if (StreamManager.findShortName(u).equals("")) {
              Text.setText ("Please enter a name");
              Input.setText ("");
       return true;
      } //if Database
      if (StreamManager.similarName(u,plist,false)) {
              Text.setText
               ("Sorry, someone with a similar name already exists.");
              Input.setText ("");
       return true;
      } //if similarName
      ((ConfFrame)getParent().getParent()).setUser(u);

      //all checks ok; continue
      return super.handleEvent(evt);
     }  //if target.equals OKButton
     if (target.equals(CancelButton)) {
      dprint ("Cancel pressed in NameWindow");
//       Client.stop();
      Client.destroy();
      return true;
     }  //if target.equals CancelButton
    } //if evt.id
    return super.handleEvent(evt);
  }  //handleEvent

}      //NameWindow

/*************************************************************/

class NewPwdWindow extends PromptWindow {
 //used for new users; asks them to choose their password
  TextField Input2 = new TextField(12);
  Prompter PromptPanel2 = new Prompter("Reenter password:",Input2);
  public NewPwdWindow (StreamManager out)
  {
    super ("New Password",
      new LabelSet("We have no record of you. If you are a previous user,",
        "please enter the same name as you used before."),
      "Choose Password:",sc.CREATEPERSTRING, out);

/*  //move buttonbar
    remove(buttonbar);
    buttonbar.removeAll();
    buttonbar.setLayout (new GridLayout(2,1));
    buttonbar.add (OKButton);
    buttonbar.add (CancelButton);
    add("East",buttonbar);
*/

    CenterPanel.setLayout(new GridLayout(3,1));
    CenterPanel.add (Text);
    CenterPanel.add(PromptPanel);
    CenterPanel.add(PromptPanel2);

  // setBackground(Color.white);
//   setForeground(Color.black);
//   Input.setBackground (Color.black);
    Input.setForeground (PROMPTBKCOLOR);   //make password invisible
//   Input2.setBackground (PROMPTBKCOLOR);
    Input2.setForeground (PROMPTBKCOLOR);
  } //NewPwdWindow constructor

  public boolean handleEvent (Event evt)  {
    if (evt.id == Event.ACTION_EVENT) {
      Object target = evt.target;
      if (target.equals(OKButton)) {
        String pwd = Input.getText();
        if (pwd.equals(Input2.getText())) {
          if (!pwd.equals("")) {
//            ((PromptFrame)getParent().getParent().getParent()).inDatabase=true;
            return super.handleEvent(evt);
        } //if !pwd
        Text.setText("You must choose a password");
      } //if pwd.equals
      else Text.setText ("The two passwords do not match.");
          Input.setText ("");
          Input2.setText ("");
      return true;
    }  //if target.equals OKButton
    if (target.equals(CancelButton)) {
      dprint("Cancel pressed in NewPwdWindow");
      Desktop d = (Desktop)getParent();
      d.removeWindow(this);
      ((ConfFrame)d.getParent()).getName
            ("Ok, let's try this again.");
      return true;
    }  //if target.equals CancelButton
  }      //if evt.id
  return super.handleEvent (evt);
 }       //handleEvent

 public Dimension size() {
  dprint ("Getting size of NewPwdWindow");

  return Display.sizeDims (Text.size(), PromptPanel.size(),
                    PromptPanel2.size(), titlebar.stringSize(), buttonbar.size());
 } //size()
}       //NewPwdWindow

/************************************************************/

class AskPwdWindow extends PromptWindow {
 String Pwd;

 public AskPwdWindow (String pwd, StreamManager out)
 {
  super ("Enter Password",
   new LabelSet("Welcome back to the conference!",
          "If you are a new user, please choose a different name.  We ",
     "already have records for someone under this name."),
   "Enter Password:", sc.CREATEPERSTRING, out);
  dprint ("AskPwdWindow: init");
  Pwd = pwd;

//   setForeground(Color.black);
  CenterPanel.setLayout(new GridLayout(2,1));
  CenterPanel.add (Text);
  CenterPanel.add(PromptPanel);

//   setBackground(Color.white);
  Input.setForeground (PROMPTBKCOLOR);  //make password invisible
  } //AskPwdWindow constructor

  public boolean handleEvent (Event evt) {
   if (evt.id == Event.ACTION_EVENT) {
     Object target = evt.target;
     if (target.equals(OKButton)) {
       dprint("OK pressed in AskPwdWindow");
       if (Input.getText().equals(Pwd)) {
//         ((PromptFrame)getParent().getParent().getParent()).inDatabase = true;
         return super.handleEvent(evt);
       } //if Input.getText
       Text.setText("Sorry, that password is incorrect", "Please try again.");
       Input.setText ("");
       return true;
     }  //if target.equals OkButton
     if (target.equals(CancelButton)) {
       dprint("Cancel pressed in AskPwdWindow");
       Desktop d = (Desktop)getParent();
       d.removeWindow(this);
       ((ConfFrame)d.getParent()).getName
             ("Ok, let's try this again.");
       return true;
     }  //if target.equals CancelButton
   }    //if evt.id
   return super.handleEvent (evt);
  }      //handleEvent
}          //AskPwdWindow

/*******************************************************************/
class NewPicWin extends PromptWindow {
//prompts user for info on importing new picture or 3-d object
  private TextField URLInput = new TextField(20);
  private Prompter URLPromptPanel = new Prompter("URL address:",URLInput);
  private String ObjType;  //type of object being imported -- image or
                    //3-d object

  public NewPicWin (StreamManager out, String otype)
  {
   super ("Import "+
      (otype.equals(sc.PICWINSTRING)?"Image":"3-D Object"),
      new LabelSet(), "Title:", (String)null, out);

   dprint ("NewPicWin: init");
   ObjType = otype;

//  URLInput.setBackground (PROMPTBKCOLOR);
//  URLInput.setForeground (PROMPTTEXTCOLOR);

   CenterPanel.setLayout(new GridLayout(3,1));
   CenterPanel.add(Text);
   CenterPanel.add(PromptPanel);
   CenterPanel.add(URLPromptPanel);
  }

 public boolean handleEvent(Event evt)
  {
   if (evt.id == Event.ACTION_EVENT) {
     if (evt.target.equals(OKButton)) {
       dprint("OK pressed in NewPicWindow");
       String title = Input.getText();
       String url = URLInput.getText();
       if (((ConfFrame)getParent().getParent()).hasImport (title,url)) {
         //import with same title or url already exists
         dprint ("preexistent picturewindow");
         Text.setText ("There is already a window open with that",
                             "title or that url address");
       } //if getParent
       else try {
         new URL(url);    //will throws MalformedURLException if bad
         if (StreamManager.findShortName(title).equals("")) {
               Text.setText ("Please enter a title");
                  Input.setText ("");
                  return true;
             }  //if Database
         Desktop d=(Desktop)getParent();
         d.removeWindow(this);
         dprint ("Valid address for picture");
                ConfFrame f=(ConfFrame)d.getParent();
         Output.send(ObjType,f.getUser(),f.getDisc(),title,url);
       } catch (MalformedURLException e) {
         dprint ("invalid URL for picture");
         Text.setText("Sorry, that was an invalid address.",
                             "Please try again.");
       }  //try/catch
       return true;
     }      //if target.equals OKButton
   }     //if evt.id
   return super.handleEvent(evt);
  } //handleEvent

  public Dimension size()
  {
   return Display.sizeDims (Text.size(),PromptPanel.size(),
          URLPromptPanel.size(), titlebar.stringSize(), buttonbar.size());
  } // size()
} //NewPicWin
/************************************************************/
class RemovePrompt extends PromptWindow {
//used to remove imports from worlds. This includes comments,
//pictures, and other imports. Each type of import is divided
```

```
//up into a list; so there are multiple lists, but the user
//can only choose one item total out of the three lists.

//CONSTANTS
 static final String NONE = "None";

//FIELDS
 Choice choices = new Choice();
 String removedType;

 public RemovePrompt (String remType,
        Enumeration remList, StreamManager out)
 //remList = list of objects that can be removed
 //remType = type, in plain English s), of object
 //      being removed
 {
  super ("Remove "+
    //simulate if then else block
    //needed because super() must be called in first line
    //of constructor
    (remType.equals(sc.COMMENTSTRING)?"Comment":
    remType.equals(sc.PICWINSTRING)?"Image":
    remType.equals(sc.OBJSTRING)?"VRML Object":"???"),
      new LabelSet ("Please choose something to remove"),
      "",sc.REMOVEIMPORT,out);
  dprint ("NewHistWin:init");

  removedType = remType;
  //create choice list
  Panel prompter = new Panel();  //contains comments
  prompter.add(choices);

  //add choices to choice menus
  choices.addItem (NONE);
  while (remList.hasMoreElements())
    choices.addItem((String)remList.nextElement());


  //adding components to CenterPanel
  CenterPanel.setLayout(new GridLayout(2,1));
  CenterPanel.add (Text);
  CenterPanel.add(prompter);
 } //NewHistWin constructor

 public boolean handleEvent (Event evt) {
   if (evt.id == Event.ACTION_EVENT) {
    if (evt.target.equals(OKButton)) {
     dprint ("handling OK button for NewHistWin");

     //determining which object is chosen
```

```
        String removed = choices.getSelectedItem();
        if (removed.equals(NONE)) {
          //nothing was selected
          Text.setText ("Sorry, you must choose something.");
          return true;
        }    //if removed

        //everything's ok; proceed
        Output.send (MessageCode,removedType,removed);
        ((Desktop)getParent()).removeWindow(this);
        return true;
      }      //if target.equals OKButton
    }     //if evt.id
    return super.handleEvent(evt);
  }        //handleEvent

/* before, removal prompt was all combined into one window
  public boolean action (Event evt, Object clicked)
  //called when user makes a choice
  //this procedure makes sure there is only one object
  //   selected at any given time
  //if the user selects anything
  //returns true if event is handled
  {
    try {
      Choice list = (Choice)evt.target;
    } catch (ClassCastException e) {
      //clicked is not a choice; let someone else handle it
      return super.action(evt,clicked);
    }

    dprint ("handling action: "+clicked);
    //go through all lists and set them to NONE, unless
    //its value is the item that was just seleced
    if (!commchoice.getSelectedItem().equals(clicked))
       commchoice.select(NONE);
    if (!imagechoice.getSelectedItem().equals(clicked))
       imagechoice.select(NONE);
    if (!vrmlchoice.getSelectedItem().equals(clicked))
       vrmlchoice.select(NONE);
    return true;
  }    //action
*/

  public Dimension size()
  {return new Dimension (250,100);}
}     //NewHistWin

/*************************************************************/
```

```
class NewDiscWin extends PromptWindow {
//CONSTANTS
  private static final String DEFAULTSIZE = "10";  //default size of
                                          //discussion
//FIELDS
//used to prompt user for title of discussion
  private TextArea DescriptInput = new TextArea ("",3,30);
  private LabelSet DescriptPrompt = new LabelSet ("Description:");
  private TextField MaxInput = new TextField (DEFAULTSIZE,3);
  Prompter MaxPanel = new Prompter ("Discussion capacity:",MaxInput);
  private Vector disclist;  //list of discussion already created -- used
    //to make sure user doesn't choose same name for discussion

  public NewDiscWin (StreamManager out, Vector dl)
  {
    super ("Create Subdiscussion",new LabelSet(""), "Topic:",
                    (String)null, out);
    disclist = dl;
    init ("");
  }

/*
  public NewDiscWin (ConfFrame parent, String descript, StreamManager out)
  //only used if topic name was already taken
  {
    super ((ConfFrame)parent, "Create Subdiscussion",
      new LabelSet ("Sorry, but the name of the discussion you",
          "tried to create has already been used",
          "Please use another name for your discussion"),
      "Topic:", (String)null, out);
    init(descript);
  }  //NewDiscWin(String,StreamManager)
*/

  private void init(String descript)
  //called by constructors
  {
    dprint ("NewDiscWin: init");
    Panel DescriptPanel = new Panel();
//   Panel MaxPanel = new Panel();
//   DescriptInput = new TextArea (descript,2,30);

    DescriptInput.setBackground (PROMPTBKCOLOR);
    DescriptInput.setForeground (PROMPTTEXTCOLOR);
    DescriptPanel.add (DescriptPrompt);
    DescriptPanel.add (DescriptInput);

//   MaxInput.setBackground (PROMPTBKCOLOR);
//   MaxInput.setForeground (PROMPTTEXTCOLOR);
//   setForeground(Color.black);
/*
   CenterPanel.setLayout(new GridLayout(4,1));
   CenterPanel.add(Text);
   CenterPanel.add(PromptPanel);
   CenterPanel.add(DescriptPanel);
   CenterPanel.add(MaxPanel);
*/
   GridBagLayout layout = new GridBagLayout();
   CenterPanel.setLayout (layout);
   GridBagConstraints c = new GridBagConstraints();
   c.fill = c.BOTH;
   c.gridwidth = c.REMAINDER;
   c.gridheight = 1;
   c.weightx = 1;  //all extra padding should go into this component
   c.anchor = c.SOUTH;  //all padding will go above the actual text
   layout.setConstraints (Text,c);
   CenterPanel.add(Text);
   c.weightx = 0;
   layout.setConstraints (PromptPanel,c);
   CenterPanel.add(PromptPanel);
   c.gridheight = 3;  //DescriptPanel is 3 times as tall as other components
   layout.setConstraints (DescriptPanel,c);
   CenterPanel.add (DescriptPanel);
   c.gridheight = 1;
   layout.setConstraints (MaxPanel,c);
   CenterPanel.add (MaxPanel);
  }   //init

  public boolean handleEvent(Event evt)
  {
   if (evt.id == Event.ACTION_EVENT) {
    if (evt.target.equals(OKButton)) {
     dprint("OK pressed in NewDiscWin");
     try {

      //checking whether maximum size of discussion is valid
      int max = Integer.parseInt(MaxInput.getText());
            if (max<2 || max>StringCode.MAXDISCSIZE) Integer.parseInt("quit");
        //throws NumberFormatException

      //reading discussion name & description
            String discname = Input.getText();
      String descript = DescriptInput.getText();
            if (descript.equals("")) descript = sc.EMPTYFIELD;

            //making sure discussion has been named
            if (StreamManager.findShortName(discname).equals("")) {
              Text.setText ("Please choose a topic name.");
              Input.setText ("");
```

```
                return true;
              }     //if StreamManager.findShortName

        //make sure discussion's name doesn't begin with
        //ConfFrame.CURDISCHEADER
        if (discname.startsWith(ConfFrame.CURDISCHEADER)) {
          Text.setText ("Sorry, you are not allowed to begin the",
                "discussion name with "+ConfFrame.CURDISCHEADER+'.');
          Input.setText ("");
          return true;
        }   //if discname

              //checking whether discussion name has been taken already
        if (StreamManager.similarName (discname,disclist,true)) {
          Text.setText ("Sorry, but a discussion with a similar",
             "or identical name has already been created.");
          Input.setText ("");
          return true;
        }   //if StreamManager.similarName

              //all checks ok; removing window, sending message
        Desktop d=(Desktop)getParent();
        d.removeWindow(this);
              ConfFrame f = (ConfFrame)d.getParent();
        Output.send(sc.NEWDISCSTRING,f.getUser(),discname,descript,
                    f.getDisc(),Integer.toString(max));
      } catch (NumberFormatException e) {
        Text.setText("Please enter a valid capacity",
           "between 2 and "+StringCode.MAXDISCSIZE);
        MaxInput.setText (DEFAULTSIZE);
      }   //try/catch
      return true;
    }       //if target.equals OKButton
   }     //if evt.id
   return super.handleEvent(evt);
  }  //handleEvent

  public Dimension size()
  {
   return new Dimension (300,200);
  }  //size()
}   //NewDiscWin

/*********************************************************************/
class NewHistWin extends PromptWindow {
 //used to open new history windows

//CONSTANTS
  static final String WILDCARD = "All";  //chosen by user if he wants
              //to view histories of any discussion or user
```

```
//FIELDS
  Choice discchoice = new Choice();
  Choice perschoice = new Choice();

  public NewHistWin (Vector discs, Vector people,
               StreamManager out)
  {
    super ("View History",
         new LabelSet ("Please choose the person and discussion",
                   "of the history you wish to view"),
         "",sc.HISTWINSTRING,out);
    dprint ("NewHistWin:init");
    Panel discprompter = new Panel();  //asks for discussion and person
    Panel persprompter = new Panel();  //for which histwin should be created

//   setForeground(Color.black);

    //add choices to choice menus
    discchoice.addItem (WILDCARD);
    perschoice.addItem (WILDCARD);
    for (Enumeration enum=discs.elements();enum.hasMoreElements();
      discchoice.addItem((String)enum.nextElement()));
    for (Enumeration enum=people.elements();enum.hasMoreElements();
      perschoice.addItem((String)enum.nextElement()));

    //creating discprompter
    discprompter.setLayout(new GridLayout(1,2));
    discprompter.add(new Label ("Discussion:"));
    discchoice.setBackground (PROMPTBKCOLOR);
    discchoice.setForeground (PROMPTTEXTCOLOR);
    discprompter.add (discchoice);

    //creating persprompter
    persprompter.setLayout(new GridLayout(1,2));
    persprompter.add(new Label ("User:"));
    perschoice.setBackground (PROMPTBKCOLOR);
    perschoice.setForeground (PROMPTTEXTCOLOR);
    persprompter.add (perschoice);

    //adding components to CenterPanel
    CenterPanel.setLayout(new GridLayout(3,1));
    CenterPanel.add (Text);
    CenterPanel.add(discprompter);
    CenterPanel.add(persprompter);
  } //NewHistWin constructor

  public boolean handleEvent (Event evt) {
    if (evt.id == Event.ACTION_EVENT) {
      if (evt.target.equals(OKButton)) {
          dprint ("handling OK button for NewHistWin");
          String disc = discchoice.getSelectedItem();
          String pers = perschoice.getSelectedItem();
          if (disc.equals(WILDCARD) && pers.equals (WILDCARD)) {
            Text.setText ("Sorry, you must choose at least a discussion",
                    "or user.");
            return true;
           }

          //everything's ok; proceed
          if (disc.equals(WILDCARD)) disc = sc.EMPTYFIELD;
          if (pers.equals(WILDCARD)) pers = sc.EMPTYFIELD;
          Output.send (MessageCode,((ConfFrame)getParent().getParent()).getUser(),
              disc,pers);
          ((Desktop)getParent()).removeWindow(this);
          return true;
        }      //if target.equals OKButton
      }     //if evt.id
      return super.handleEvent(evt);
  }        //handleEvent

  public Dimension size()
  {return new Dimension (300,120);}
}      //NewHistWin

/*********************************************************************/

class Prompter extends Panel {
//involves 2 components: a prompt, and a textfield into which the user inputs
  public static final boolean DEBUG = Display.DEBUG;
  LabelSet prompt;
  TextField input;

  public Prompter (String promptstr, TextField i)
  {

    add (prompt=new LabelSet(promptstr));
    add(input=i);
    input.setBackground (PromptWindow.PROMPTBKCOLOR);
    input.setForeground (PromptWindow.PROMPTTEXTCOLOR);
  }

  public Dimension size()
  {
    Dimension psize = prompt.size();
    int iwidth = getGraphics().getFontMetrics().stringWidth("m")*
          input.getColumns();
    Display d = new Display();

    if (DEBUG) System.out.println ("Prompter size:"+
                        (psize.width+iwidth+3*d.MARGIN)+','+
                        (psize.height+d.TEXTMARGIN+2*d.MARGIN));

    return new Dimension (psize.width+iwidth+3*d.MARGIN+20,
                        psize.height+d.TEXTMARGIN+2*d.MARGIN);
  }     //size()
} //Prompter
```

---

```
/*ScaleConstraint.java
contains ScaleConstraint class.

This class holds layout information of a particular window.  it is used
by ScaleLayout (see ScaleLayout.java).
*/

package WinPack;
import java.awt.*;
import java.util.*;

class ScaleConstraint
// Structure that holds all the layout information for a component
{

// FIELDS
  public int x;
  public int y;
  public int width;
  public int height;

// METHODS
  public ScaleConstraint () { }

  public ScaleConstraint(int px, int py, int w, int h)
  {
    x = px;
    y = py;
    width = w;
    height = h;
  } // end ScaleConstraint constructor
} // end ScaleConstraint
```

---

```
/*ScaleLayout.java
Contains ScaleLayout class.

This class is used by the Desktop (see Desktop.java) to handle the layout
for windows.  It contains a list of all the windows on the desktop, and
the corresponding ScaleConstraints (see ScaleConstraint.java).  It allows
windows to be moved anywhere in the desktop, but keeps the sizes of the
windows constant.
*/
```

```
package WinPack;
import java.awt.*;
import java.util.*;

class ScaleLayout implements LayoutManager
{
//CONSTANTS
 private final static boolean DEBUG = Desktop.DEBUG;  //DK 7/31

// FIELDS
 Dimension d;
 Hashtable Constraints = new Hashtable (10);

// METHODS
 public  ScaleLayout(Dimension dim)
 {
  d = dim;
 } // end ScaleLayout constructor

 public void addLayoutComponent(String name, Component comp)
 // effects: adds comp to this layout
 {
  if (DEBUG) System.out.println ("adding component");
  Constraints.put (comp,new ScaleConstraint());
 } // end addLayoutComponent

 public void removeLayoutComponent(Component comp)
 // effects: removes comp from this layout
 {Constraints.remove (comp);}

 public Dimension minimumLayoutSize(Container target)
 {return d;}

 public Dimension preferredLayoutSize(Container target)
 {return d;}

 public void setConstraint (Component self, int px, int py)
 //used once window is moved
 {
  ScaleConstraint con = (ScaleConstraint) Constraints.get(self);
  con.x = px;
  con.y = py;
  plopComponent (self.getParent(),self);
 }

 public void setConstraint(Component self, int px, int py, int w, int h)
 // effects: sets new layout information for self
 {
  if (DEBUG) System.out.println ("setting constraints: "+
         px+','+py+','+w+','+h);
  Constraints.put (self,new ScaleConstraint (px,py,w,h));
  if (!plopComponent (self.getParent(),self))
    System.err.println("Error:trying to plop component without constraints");
 } // end setConstraint

 public void layoutContainer(Container target)
 // effects: lays out all components in target (the container that has this layout)
 {
  for (Enumeration e = Constraints.keys(); e.hasMoreElements();
         layoutComponent (target, (Component)e.nextElement()));
 } // end layoutContainer

 public void layoutComponent(Container target, Component c)
 // effects: lays out c in target with information contained in sc
 {
  ScaleConstraint sc = (ScaleConstraint)Constraints.get(c);

  if (DEBUG) System.out.println("laying out component:"+sc.x+','+sc.y+','+
                                              sc.width+','+sc.height);
  c.move(sc.x, sc.y);
  c.resize(sc.width, sc.height);
 }  // end layoutComponent

 public boolean plopComponent(Container target, Component c)
 // effect: more ropust way to lay out component:  makes sure that
 // constraint information is updated
 {
  if (Constraints.containsKey(c)) {
    layoutComponent (target, c);
    return true;
   } return false;
 } // end plopComponent

 public ScaleConstraint getConstraint (Component c)
 //DK 7/30
 //returns constraints for c
 {
  if (DEBUG) System.out.println ("getting constraint for window: "+c);
  return (ScaleConstraint)Constraints.get(c);
 }  //getConstraint
}
```

---

```
/* SendWindow.java
Contains SendWindow, SendCommWindow and SendChatWindow classes.

SendWindow is the parent class of the other two classes.
SendChatWindow is used to chat.  When a user opens a SendChatWindow, a
   ChatWindow (see TalkWindow.java) appears on other user's desktops.
   Any text the user sends through his SendChatWindow will appear on the
   other user's ChatWindow.
```

```
SendCommWindow is used for posting comments.  Once a user hits the "Send"
  button, it closes, and places the new comment in the 3-d World (after
  being processed by the server)
*/

package WinPack;
import java.awt.*;
import java.util.*;
import StreamManager;

abstract class SendWindow extends EmbeddedCenteredWindow
//Window used to display history
{
// FIELDS
 protected StreamManager Output;
 public TextArea display;
 protected Button sendbutton = new Button ("Send");
 protected Button clearbutton = new Button ("Clear");
 protected Button closebutton= new Button ("Close");

// PUBLIC METHODS
 public SendWindow(StreamManager o, String title,
                    boolean modal)
 {
  super (title);
  if (DEBUG) System.out.println ("SendWindow: init");
  Output = o;
  display = new TextArea();
  display.requestFocus();  /*doesn't work*/
  add("Center",display);
  buttonbar.add(sendbutton);
  buttonbar.add(clearbutton);
  buttonbar.add(closebutton);
 } // end init

 public boolean handleEvent(Event evt)
 {
  if (evt.id == Event.ACTION_EVENT) {
   if (evt.target.equals(clearbutton)) {
    display.setText ("");
    return true;
   }  //if target.equals clearbutton
  }  // if evt.id
  return super.handleEvent(evt);
 }  //handleEvent

 public String identifiers() {return sc.EMPTYFIELD;}

 public boolean MulWinsAllowed() {return true;}
}  //SendWindow
```

```
/***************************************************************/

class SendChatWindow extends SendWindow {
  public SendChatWindow (StreamManager o)
  {
    super(o, "Chatting Window", false);
    if (DEBUG) System.out.println ("SendChatWindow:init");
  }

  public boolean handleEvent(Event evt)
  {
    if (evt.id == Event.ACTION_EVENT) {
      Object target = evt.target;
      if (target.equals(sendbutton)) {
              String text = display.getText();
        if (!text.equals(""))
          Output.send (sc.SETTEXT,
          ((ConfFrame)getParent().getParent()).getUser(), text);
        return true;
      }   //if target.equals sendbutton

      if (target.equals(closebutton)) {
        Desktop d=(Desktop)getParent();
        d.removeWindow(this);
        ConfFrame f=(ConfFrame)d.getParent();
        if (d.findWindow(sc.SENDCHATSTRING,sc.EMPTYFIELD)==null)
                Output.send (sc.CLOSECHAT,f.getUser());
        return true;
      }     //if evt.target
    }  //if evt.id
    return super.handleEvent(evt);
  } // end handleEvent

  public String Type() {return sc.SENDCHATSTRING;}
}         //SendChatWindow

/***************************************************************/

class SendCommWindow extends SendWindow {
  public SendCommWindow (StreamManager o)
  {
    super(o,"Post a comment", true);
    if (DEBUG) System.out.println ("SendCommWindow:init");
  }  //SendCommWindow constructor

  public boolean handleEvent(Event evt)
  {
    if (evt.id == Event.ACTION_EVENT) {
      Object target = evt.target;
    if (target.equals(sendbutton)) {
      String text = display.getText();
      if (!text.equals (""))
        Output.send (sc.COMMENTSTRING,
          ((ConfFrame)getParent().getParent()).getUser(), text);
  //no return statement so that next if statement is checked
    }  //if evt.target
    if (target.equals(closebutton)||target.equals(sendbutton)) {
      Desktop d=(Desktop)getParent();
      d.removeWindow(this);
      return true;
    }     //if evt.target
  } //if evt.id
  return super.handleEvent(evt);
 } // end handleEvent

 public String Type() {return sc.SENDCOMMSTRING;}
} //SendCommWindow
```

---

```
/* TalkWindow.java
contains TalkWindow and ChatWindow classes

TalkWindows are created as a counterpart to SendWindows
   (see SendWindow.java).  It used to have 2 subclasses (in GTT):
   ChatWindow and CommentWindow, but now that comments are included in
   the VRML world in DDS, it only has one (ChatWindow).
ChatWindow is the counterpart of SendChatWindow (see SendWindow.java).  It
   is created when another user opens a ChatWindow on this desktop.
*/

package WinPack;
import java.awt.*;
import java.util.*;
import StreamManager;
import java.net.URL;
//import EmbeddedCommentWindow;

abstract class TalkWindow extends EmbeddedLeftJustifiedWindow
{
// FIELDS
 public Component display;
 public String Speaker;
 protected StreamManager output;  //DK 7/20
 protected String Disc;     //DK 7/22

// METHODS
 public TalkWindow (String title, String disc,
                 int dwidth, int dheight, StreamManager out)
 //DK; first created 7/7;  last update 7/13
 {
  super(title);
  System.out.println ("TalkWindow:init");
  Disc = disc;
  output = out;
 }        //TalkWindow constructor

 public boolean handleEvent(Event evt)
 {
  if (evt.id == Event.ACTION_EVENT) {
    Object target = evt.target;
    Desktop d = (Desktop)getParent();
  }  //if evt.id
  return super.handleEvent(evt);
 } // end handleEvent

 public String identifiers() {return Speaker;}

 public boolean MulWinsAllowed() {return false;}
} // end TalkWindow
/*******************************************************/

class ChatWindow extends TalkWindow{
//windows corresponding to SendChatWindows

 public ChatWindow (String spkr, String text, String disc,
                    int dwidth, int dheight, StreamManager out)
 {
  super ("Chat: "+spkr,disc,dwidth,dheight,out);
  display = new Display ("", dwidth, dheight);
  ((Display)display).setEditable(false);
  ((Display)display).setText (text);
  add("Center", display);
  Speaker = spkr;
 }     //ChatWindow constructor

 public void setText(String s)
 //sets text of display to s
 {
  if (DEBUG) System.out.println ("setting text for TalkWindow to "+s);
  ((Display)display).setText(s);
 }

 public String identifiers() {return Speaker;}

 public String Type() {return sc.CHATSTRING;}
}   //ChatWindow
```

---

```
/*TitleBar.java
Contains TitleBar class.
```

```
This class is at the top of every EmbeddedWindow.  When the user clicks on
   it and drags the mouse, he can move the window.
*/

package WinPack;
import java.awt.*;
import java.util.*;
import java.net.*;

class TitleBar extends Canvas
// This is the class that allows dragging on the Desktop
// NOTE:  There is probably a cleaner way to do this.  The Desktop
// class should be rewritten to make dragging easier, or a generic
// Titlebar widget should be implemented.
{
// CONSTANTS
  public static final boolean DEBUG = Display.DEBUG;
// FIELDS
  public String title;
  Dimension ss;
  int string_leading, string_ascent;

// METHODS
  public TitleBar(String t)
  {
    title = t;
    setFont(ConfFrame.titleFont);
    setForeground(Color.black);
    setBackground(Color.white);
    measureString();
    resize(ss.width, ss.height);
    System.out.println("Title Bar Size: " + ss);
  } // end TitleBar constructor

  public void paint(Graphics g)
  // effects: draws titlebar with title centered.
  {
    int width = size().width;
   g.drawString(title, (int)((width - ss.width)/2),
                              Display.MARGIN + string_leading / 2 + string_ascent);
  } // end paint

  public void setText (String text)
  {
    title = text;
    repaint();
  }

  public Dimension stringSize()
  {return ss;}
  private void measureString()
  {
   FontMetrics f = Toolkit.getDefaultToolkit().getFontMetrics(getFont());
   if (DEBUG) System.out.println ("title size("+title+"):"+
                                  (f.stringWidth (title)+2*Display.MARGIN)+','+
                                  (f.getHeight()+2*Display.MARGIN));
   ss = new Dimension (f.stringWidth (title)+2*Display.MARGIN,
                                  f.getAscent() + f.getDescent() + f.getLeading() + 2*Display.MARGIN);
   string_ascent = f.getAscent();
   string_leading = f.getLeading();
  }

  public boolean handleEvent(Event evt)
  {
   switch(evt.id)
   {
    case Event.MOUSE_DOWN:
    {
     Container par = getParent();
         try {
      ((Desktop)(par.getParent())).select(par, evt.x, evt.y);
     } catch (ClassCastException e) { }
     return true;
    }
    default:
         return false;
   } //switch evt.id
  } // end handleEvent

  public Dimension minimumSize()
  {return ss;}

  public Dimension preferredSize()
  {return ss;}

} // end TitleBar
```

---

```
/*WinDefault.java
Contains class WinDefault only.

This class is a data structure that contains information about
the size and position of imports in a discussion. The name of
the class originated in GDT, when all imports were contained
in windows. However, now, WinDefault is used by imports
of all kinds, including 3-D imports, images, and comments.
*/
package WinPack;
import java.util.StringTokenizer;
import StringCode;
```

```
class WinDefault {
//written by DK 7/26
//stores defaults for windows
//makes up ClosedWinLayout
  private static final StringCode sc = new StringCode();
  private static final boolean DEBUG = EmbeddedWindow.DEBUG;

  public String WinType;  //type of window, using StringCode
  public String Identifiers; //to identify the windows, corresponds to
                      //"params" variable in EmbeddedWindow.matches
  int x;  int y;  //coordinates of window
  int w;  int h;  //dimensions of window

  public WinDefault (String params) {
    if (DEBUG) System.out.println ("creating WinDefault: "+params);
    StringTokenizer tok = new StringTokenizer (params, sc.PARAMDIVIDER, false);
    WinType = tok.nextToken();
    Identifiers = tok.nextToken();
    x = Integer.parseInt (tok.nextToken());
    y = Integer.parseInt (tok.nextToken());
    w = Integer.parseInt (tok.nextToken());
    h = Integer.parseInt (tok.nextToken());
  }  //WinDefault constructor with StringTokenizer

  public WinDefault (String wintype, String id, ScaleConstraint constraint) {
    if (DEBUG) System.out.println ("creating WinDefault");
    WinType = wintype; Identifiers = id;
    x=constraint.x; y=constraint.y; w=constraint.width; h=constraint.height;
  }  //WinDefault constructor using ScaleConstraint

  public String toString() {
    return WinType+sc.PARAMDIVIDER+Identifiers+sc.PARAMDIVIDER+x+
          sc.PARAMDIVIDER+y+sc.PARAMDIVIDER+w+sc.PARAMDIVIDER+h;
  } //toString
} //WinDefault
```

## III. VRML World Frame

```
/* Comment.java
includes Comment class

This class is a representation of the comment windows in the 3-D world.
*/

import vrml.*;
import vrml.field.*;
import vrml.node.*;
import java.util.StringTokenizer;
import java.util.NoSuchElementException;
import Import;

public class Comment extends Import {
//CONSTANTS
 private final static float COMMENTWIDTHRATIO = (float).1;
  //number to multiply length of longest line by to get width
  //of comment window for VRML world
 private final static float COMMENTHEIGHTRATIO = (float).15;
  //number to multiply number of lines of text by to get
  //height of text are of comment window
 private final static int TEXTRANGE = 15; //farthest distance for text

//FIELDS
// float commentWidth;       //dimensions of Comment, without
// float commentHeight;      //title bar

//METHODS
 public Comment (String spkr, String comment,
  float x, float y, float z, Browser b, MFNode addObj,
  SFNode cl)
 throws InvalidVRMLSyntaxException
 {super(spkr,comment,x,y,z,b,addObj,cl);}

 protected Node[] createObj (String spkr, String text,
       Browser browser, float dims[])
 throws InvalidVRMLSyntaxException
 //creates comment window with given speaker and comment
 {
  //getting size of text & parsing it
  int numlines = countLines (text); //number of lines in text
  int longestline = findLongest(text); //number of characters in the
              //longest line
  String parsedQuote = parseText(text);

  //calculating size of area in comment window with text
  dims[X] = dims[Z] = longestline*COMMENTWIDTHRATIO+2*MARGIN;
    //This number is an estimate, because all characters are actually
        //different widths
  dims[Y] = numlines*COMMENTHEIGHTRATIO+2*MARGIN;
  float[] commentTextPos = {-dims[X]+MARGIN,-dims[Y]+MARGIN};
       //position, relative to center of comment window, at which to
       //start drawing text (upper left corner)

  //creating node
  String commstr =
   "Billboard {\n"+     //comment area of window
   "  children [\n"+
   "    Shape {\n"+
    "     appearance Appearance {\n"+
    "       material Material { diffuseColor 0 0 0 }\n"+
    "      }\n"+      //appearance
    "     geometry Box { size "+dims[X]+' '+dims[Y]+" 0 }\n"+
    "    }\n"+ //Shape
    "   LOD {\n"+ //text is only visible within certain range
    "     range "+TEXTRANGE+'\n'+
    "     level [\n"+
    "       Transform {\n"+                         //text for comment
    "         translation "+commentTextPos[0]+' '+
                commentTextPos[1]+' '+TEXTPROTRUSION+'\n'+
    "         children Shape {\n"+
    "           appearance Appearance {\n"+
    "             material Material { diffuseColor 1 1 1 }\n"+
    "           }\n"+  //Appearance
    "           geometry Text {\n"+
    "             string "+parsedQuote+"\n"+
    "             fontStyle FontStyle {\n"+
    "               size .15\n"+
    "               family \"SANS\"\n"+
    "               style \"BOLD\"\n"+
    "             }\n"+   //fontStyle
    "           }\n"+   //geometry Text
    "         }\n"+   //Shape
    "       },\n"+ //Transform
    "       Group { }\n"+
    "     ]\n"+ //level
    "   }\n"+ //LOD
    "  ]\n"+ //children
    '}'; //Transform

  //create the actual nodes
  dprint ("creating comment from string:"+commstr);
  Node [] commnode =
    (Node[])browser.createVrmlFromString (commstr);
  Node[] titlenode = createTitleNode(spkr, dims[X],
                           dims[Y], browser);
  Node[] returnnode = {commnode[0],titlenode[0]};
  return returnnode;
} //createObj

protected String viewpointDescription (String title)
//description of viewpoint for this object in the VRML Browser
{return "Comment: "+title;}

//PRIVATE METHODS
private int countLines (String text)
//counts number of lines (separated by \n) in text
{return (new StringTokenizer (text,"\n",false)).countTokens();}
```

```
private int findLongest (String text)
//finds number of characters in longest line of text
//each line is assumed to be separated by a newline
{
 StringTokenizer tok = new StringTokenizer (text, "\n", false);
 int longest = 0;
 try {
  int nextlinelen;
  while (true) {
   nextlinelen = tok.nextToken().length();
   if (nextlinelen>longest)
    longest = nextlinelen;
  } //while true
 } catch (NoSuchElementException e) { }    //end of text reached
 dprint ("longest line: "+longest);
 return longest;
}     //longestLine

private String parseText (String text)
//converts text into a string compatible with VRML files
//The return string follows the format of an MFString
{
 dprint ("parsing text");
 StringTokenizer tok = new StringTokenizer (text, "\n", false);

 //first couple of lines of returnstr
 StringBuffer returnstr = new StringBuffer ("[\n");
 if (tok.hasMoreTokens())
  returnstr.append("\""+tok.nextToken()+"\"");
 else return ("[]");     //empty MFString

 //go through line by line until we reach end of text
 try {
  while (true)
   returnstr.append(",\n\""+tok.nextToken()+"\"");
 } catch (NoSuchElementException e) { } //end of text reached

 //finish off returnstr
 returnstr.append("\n]");
 return returnstr.toString();
} //parseText
} //Comment
```

---

```
/*CommentProto.java
contains CommentProto class.

This class is the prototype used for comments that are added
to subdiscussion prototypes. All CommentProtos are the same
size, and they display no text. They simply show the title
bar (without text) and the text area.
```

```
*/

import vrml.*;
import vrml.node.*;
import vrml.field.*;
import ImportProto;
import StringCode;

public class CommentProto extends ImportProto {
//CONSTANTS
private static final float WIDTH = 2;  //of CommentProto
private static final float TEXTAREAHEIGHT = 1;  //height of
          //textarea
private static final float TITLEHEIGHT = (float) .5;
          //of title bar

//METHODS
public CommentProto (float x, float y, float z, Browser b,
   MFNode addObj)
throws InvalidVRMLSyntaxException
//x,y,z = position at which to place CommentProto
//addObj = eventIn used to add CommentProto. This eventIn
//  should be the eventIn of the Subdiscussion prototype
{super (StringCode.EMPTYFIELD,x,y,z,b,addObj);}

protected Node[] createObj (String url, Browser b)
  throws InvalidVRMLSyntaxException
//for CommentProto, url is ignored
{
 return (Node[])b.createVrmlFromString (
 "Group {\n"+
 "  children [\n"+
 "   Transform {\n"+     //title bar
 "    translation 0 "+((TEXTAREAHEIGHT+TITLEHEIGHT)/2.0)
          +" 0\n"+
 "    children Shape {\n"+
 "     appearance Appearance {\n"+
 "      material Material { diffuseColor 1 1 1 }\n"+
 "     }\n"+ //appearance
 "     geometry Box {\n"+
 "      size "+WIDTH+' '+TITLEHEIGHT+" 0\n"+
 "     }\n"+
 "    }\n"+ //Shape
 "   }\n"+ //Transform
 "   Transform {\n"+     //text area
 "    children Shape {\n"+
 "     appearance Appearance {\n"+
 "      material Material { diffuseColor 0 0 0 }\n"+
 "     }\n"+ //appearance
 "     geometry Box {\n"+
"       size "+WIDTH+' '+TEXTAREAHEIGHT+" 0\n"+
"      }\n"+
"    }\n"+ //Shape
"  }\n"+ //Transform
" ]\n"+ //children
'}'); //Group
}       //createObj
}       //CommentProto
```

---

```
/*ConfClient.java
Contains ConfClient and WorldInLoop classes.

ConfClient is the Script class that is started when DDS's VRML world loads
up (see DDS.wrl). It opens a connection with the WorldServer (see
ConferenceServer.java) on the server side. To handle the message sent from
the server, a WorldInLoop is created.

The client side of DDS operates in a 3-frame environment (see DDS.html):
  1) world -- contains the actual VRML world. This is the frame the
      ConfClient script is initialized & run in. All manipulations of
      the world are managed by the WorldHandler class, but responses to
      buttons are handled in ConfClient.
  2) interface -- contains HTML pages that are used for user interface.
      These pages communicate information to the server
  3) history -- displays requested histories. Currently, only one history
      can be displayed at a time
*/

import vrml.*;
import vrml.field.*;
import vrml.node.*;
import java.net.*;
import java.io.*;
import java.lang.*;
import java.util.*;
import java.awt.Dimension;
import InputLoop;
import StringCode;
import WorldHandler;

/*
 *       This class handles the communication for the client and also
 *       creates a user interface (through WorldHandler)
 */

public class ConfClient extends Script
// This class opens up a request socket to the ConferenceServer,
// and provides a graphical user interface (WorldHandler) for the discussion
{
```

```
// CONSTANTS
  public static final boolean DEBUG = true;
  public static final StringCode sc = new StringCode();

// FIELDS
  public boolean inDatabase = false;  //set true once user enters a discussion
  public Socket world_socket;
//  public Socket broadcast_socket;
  public DataInputStream world_input;
  public StreamManager world_output;
//  public DataInputStream broadcast_input;
//  public StreamManager broadcast_output;
  public WorldHandler whandler;
//  public PromptInLoop promptLoop;
  public WorldInLoop worldLoop;

// PUBLIC METHODS

  public void initialize()
  // effects:  create world socket and associate input and
  // output streams with them.  Then start world input loops
  // to constantly check for input from the server
  // Also creates initial user interface.
  {
    System.out.println ("initializing ConfClient");
    //byte bytes[] = new byte[4096];
    try {
//      broadcast_socket = new Socket(sc.HOSTNAME, sc.BROADCAST_PORT);
//      broadcast_input = new DataInputStream(broadcast_socket.getInputStream());
//      broadcast_output =
//          new StreamManager(new PrintStream(broadcast_socket.getOutputStream()));
      world_socket = new Socket(sc.HOSTNAME, sc.WORLD_PORT);
      world_input = new DataInputStream(world_socket.getInputStream());
      world_output =
          new StreamManager(new PrintStream(world_socket.getOutputStream()));
//      whandler = new WorldHandler(this, "Main","", world_output,
//        getBrowser(), (MFNode)getEventOut ("addObj"),
//        (MFNode)getEventOut ("removeObj"), (MFNode)getEventOut ("addComm"),
//        (MFNode)getEventOut ("removeComm"));
      world_output.send(sc.NEWUSERSTRING);

      //create WorldHandler
      whandler = new WorldHandler (sc.DEFAULTDISC,"",
        world_output,getBrowser(),
        (SFNode)getField ("thisNode"),
        (SFNode)getField ("world"),
//        (SFNode) getField ("commentWall"),
        (SFNode)getField ("subDiscHolder"),
        (SFBool)getEventOut("setExaminer"),
        (SFBool)getEventOut("frontView"),
      (SFColor)getEventOut("changeXColor"),
      (SFColor)getEventOut("changeYColor"),
      (SFColor)getEventOut("changeZColor"),
      (SFBool)getEventOut("enableXPSensor"),
      (SFBool)getEventOut("enableYPSensor"),
      (SFBool)getEventOut("enableZPSensor"),
      (SFNode)getField("axes"));

     worldLoop = new WorldInLoop(world_input, whandler);
     worldLoop.start();
    }
    catch(Exception e) {
     StreamManager.showError(e);
     try {
//      broadcast_socket.close();
          world_socket.close();
     }
     catch(Exception e2) {StreamManager.showError(e2);}
     System.exit(1);
    }
  } // end initialize

  public void processEvent (Event evt) {
   whandler.processEvent (evt);
  }

  public void shutdown()
  {
  // effects: destroys all threads that have been created by this applet
  // NOTE: you MUST destroy all threads that you create!
   if (DEBUG) System.out.println ("ConfClient: shutdown()");
   inDatabase = false;
//   promptLoop.destroy();
   worldLoop.stop();
  } // end destroy
} // end ConfClient

/**********************************************/

class WorldInLoop extends InputLoop
// This class constantly checks the world input stream to see if the server
// has responded to a world.
{
// FIELDS
  public WorldHandler whandler;

// METHODS
  public WorldInLoop(DataInputStream in, WorldHandler cf) {
   super(in);
   whandler = cf;
```

```
}

public boolean process_message(String server_message) {
// effects:  here is where the client deals with responses from the
// server.  To deal with a new kind of response from the server, simply
// add another "if" statement below
  if (ConfClient.DEBUG) System.out.println("processing in world:"
        + server_message);

try {
  StringTokenizer tok = StreamManager.parseMessage (server_message);
  String code = tok.nextToken();

  //FOR WINDOWS/BUTTONS
/*
  if (code.equals(sc.LAYOUTSTRING)) {
    //load layout into memory
    whandler.setWinLayout(tok.nextToken());
    return true;
  }        //if code.equals LAYOUTSTRING
*/

  if (code.equals(sc.PICWINSTRING)) {
  //place image in world
    whandler.addImage (tok.nextToken(),  //title
        tok.nextToken(),        //url
        tok.nextToken());       //coordinates
    return true;
  } //if code.equals PICWINSTRING

  if (code.equals(sc.OBJSTRING)) {
  //place 3-D object in world
    whandler.add3DObj (tok.nextToken(),  //title
        tok.nextToken(),        //url
        tok.nextToken());       //coordinates
    return true;
  } //if code.equals OBJSTRING

  if (code.equals(sc.COMMENTSTRING)) {
  //add comment to comment wall
    whandler.addComment (tok.nextToken(),  //speaker
        tok.nextToken(),        //comment
        tok.nextToken());       //coordinates
    return true;
  } //if code.equals COMMENTSTRING

  if (code.equals(sc.OBJMOVEDSTRING)) {
    //move given window on desktop
    whandler.moveImport (tok.nextToken(), //type of the object
            tok.nextToken(),        //id for moved object
            tok.nextToken());      //new position
      return true;
    }

    if (code.equals(sc.REMOVEIMPORT))
    //remove Import from world
    {
      whandler.removeImport (tok.nextToken(), //object type
                tok.nextToken()); //identifier
      return true;
    } //if code.equals REMOVEOBJ

    //FOR SWITCHING DISCUSSIONS
    if (code.equals(sc.ASKFORSIZE)) {
      //the server is requesting to update the size it has
      //recorded for this discussion's VRML world.
      //This is sent any time something new is imported or
      //removed, but only to the person who did the importing
      //or removing
      whandler.sendSize();
      return true;
    } //if code.equals ASKFORSIZE
    if (code.equals(sc.ADDSUBDISC)) {

      //create new prototype for subdiscussion
      whandler.addSubDisc (tok.nextToken(), //title
            tok.nextToken(), //lowest bounds
            tok.nextToken()); //uppermost bounds
      return true;
    }  //if code.equals ADDSUBDISC

    if (code.equals(sc.ADDSUPERDISC)) {
      //set name parent discussion
      whandler.addSuperDisc (tok.nextToken());
      return true;
    }  //if code.equals ADDSUPERDISC

    if (code.equals(sc.ADDPROTO)) {
      //adds prototype to given subdiscussion
      whandler.addProto (tok.nextToken(),
                //type of import (comment/image/etc.)
            tok.nextToken(), //name of subdisc
            tok.nextToken(), //url of prototype
            tok.nextToken()); //position
      return true;
    }     //if code.equals ADDPROTO

   if (code.equals(sc.CLEARDESKTOP)) {
//    ConfClient client = whandler.Client;
      whandler.reset (tok.nextToken());
```

```
/*    Hashtable pd = client.PrevDiscs;
    String olddisc = whandler.Disc;
    String layoutstr = client.newFrame(tok.nextToken());
    pd.put(olddisc,layoutstr);
*/
    return true;
   }  //if code.equals CLEARDESKTOP
  } catch(Exception e) {StreamManager.showError(e);}
  System.out.println ("Message not processed: "+server_message);
  return false;
} // end process_message

protected void handleSocketException ()
//called when server crashes, or unexpected disconnection
//occurs
{
  System.err.println ("World: connection lost with server");
//  super.handleSocketException();
}
   //WorldInLoop just leaves the world as is but PromptInLoop
   //shows an error message (seePromptFrame.java
} // end WorldInLoop
```

---

```
/* Import.java
contains abstract class Import.

This class is the superclass of all imports (images and 3-D objects, and
maybe more in the future), and Comments.
*/

import vrml.*;
import vrml.field.*;
import vrml.node.*;
import ObjInfo;

public abstract class Import {
//CONSTANTS
  public final static boolean DEBUG = WorldHandler.DEBUG;
  public final static int X = 0;
  public final static int Y = 1;
  public final static int Z = 2;


  //used by Comment class and Import.createTitleNode()
  protected final static float MARGIN = (float).2;
   //margin between text and
          //edge of box on which it is drawn (for title & comments)
  protected final static float TEXTPROTRUSION = (float).05;
   //distance in front of
          //boxes at which texts are drawn (for title & comments)
```

```
 //these constants used by createTitleNode()
 private final static int TITLERANGE = 20; //farthest distance at which a
          //user can see the text of the title bar in the VRML world
 private final static float TITLEWIDTHRATIO = (float).1;
      //number to multiply
      //length of title by to get width of title bar
 private final static float TITLEHEIGHTRATIO = (float).15;
   //number to multiply
          //number of lines in title by to get height of title text
          //Used to determine where to position text node.
 private final static float TITLEHEIGHT = (float).4;
   //height of title bar
 private final static float VIEWTOOBJ = (float).1;   //base
   //distance from ViewPoint to object.  Note that because
   //the size of Comments and Images is the a box, and they
   //are billboards, .1 away from the object means more than
   //.1 away from the center of the object.  So in actuality
   //the distance from the object is much more than .1
// private final static float EXTRAVIEWRATIO = .5;  //for wide
   //or tall imports, the ViewPoint should be set farther back
   //so that the user can see the whole object.  This ratio
   //is used to calculate how far back the user should be set.

//FIELDS
/*
 protected BaseNode[] obj;  //actual object that is imported -- is a Group
      //with TouchSensor and object
 private BaseNode[] script;   //script to which events related to this
            //object are routed
 private BaseNode[] touchSensor;  //Touch Sensor of this node
      //this is a child of
 protected Browser browser;  //browser in which VRML world is
*/
 protected ObjInfo info;
 Node[] script;      //TouchTranslator script
 Node[] obj;      //import, with sensors & title bar
 Node[] tSensor;  //touchSensor
// Node[] view;     //Viewpoint
 Node[] itemHolder; //holds object, title bar, and viewpoint
      //this is embedded in all the translations and PlaneSensors
 Node client;     //ConfClient Script node

//PUBLIC METHODS
 public Import (String ti, String str, float x, float y, float z,
         Browser b, MFNode addObj, SFNode cl)
 throws InvalidVRMLSyntaxException
 //creates script, touchSensor, and obj and appropriate routes
 //str = urls for images & imported 3-Ds; comment text for
 //   CommentWins
```

```
  {
   dprint ("creating import: "+ti);
   client = (Node)cl.getValue();

   //creating TouchSensor, TouchTranslator script, and route
   //so that ObjMover can figure out which object has its
   //title bar clicked
//   script = createScript(ti,b);
   tSensor = createTSensor(b);
//   b.addRoute
//      (tSensor[0],"touchTime",script[0],"touchTime");
//   b.addRoute
//      (script[0],"touched",client,"titleClicked");
//   addObj.setValue(script);
   dprint ("TouchTranslator added to world");

//   Node[] title = createTitle(ti);
   float[] dims = {0,0,0}; //allocating memory
       //the elements of the array are changed by createObj

   //creating nodes
   Node[] whole = createObj(ti, str, b, dims);
   Node[] title = {whole[1]};
   Node[] objNoSensors = {whole[0]};
//   ((MFNode)((Node)obj[0]).getEventIn("addChildren")).
//         setValue(touchSensor);
   if (objNoSensors==null)
      throw new InvalidVRMLSyntaxException();

  //because viewpoint can't be removed, it is no longer used
//   view = createViewPoint (dims,ti,b);

//   info = addSensors(title,objNoSensors,view,x,y,z,b);
   info = addSensors(title,objNoSensors,x,y,z,b);
   info.setDims(dims);

   addObj.setValue(obj);
   dprint ("import added to world");

// should be uncommented when TouchTranslators work
//   addScript.setValue(script);
//   b.addRoute (touchSensor,"touchTime",script,"touchTime");
//   b.addRoute (script,"moveObj",obj,"translation");
  } //init

  public ObjInfo getInfo()
  {return info;}

  public Node[] getNode()
  {return obj;}
  public void remove (MFNode removeObj, Browser b)
  //uses EventIn removeObj to remove object (w/ touchSensor) & removeScript
  //to remove script from world.  Also removes appropriate routes
  {
    dprint ("Import.remove() called");

    //remove viewpoint
    //removal doesn't work
//    MFNode removeFromObj = (MFNode)itemHolder[0].getEventIn("removeChildren");
//    removeFromObj.setValue(view);

    //remove actual object
    removeObj.setValue(obj);

    //remove script
//    b.deleteRoute
//       (tSensor[0],"touchTime",script[0],"touchTime");
//    b.deleteRoute
//       (script[0],"touched",client,"titleClicked");
//    removeObj.setValue(script);
  }  //remove

//PROTECTED METHODS
// protected ObjInfo addSensors (Node[] titlebar,
//       Node[]imported, Node[] viewpoint, float x, float y,
//       float z, Browser browser)
  protected ObjInfo addSensors (Node[] titlebar,
       Node[]imported, float x, float y,
       float z, Browser browser)
  throws InvalidVRMLSyntaxException
  //modifies: this.obj
  //titlebar is a Transform that already contains the correct
  //   translation for the title bar
  //imported is the imported object
  //viewpoint = viewpoint facing the object
  //x,y,z = the position at which the object should be placed
  //dims = size of the object
  //creates node with given titlebar ad imported object, with
  //   PlaneSensors and TouchSensors (on title bar).
  //adds routes from those PlaneSensors to the appropriate
  //   Nodes
  //sets obj, to resulting node (with sensors)
  //ObjInfo contains PlaneSensors and node to which to add
  //   axes for moving object
  //the info field of this class is set to this ObjInfo
  {
    dprint ("adding sensors to new import");
    Node whole =        //contains everything
       ((Node[])browser.createVrmlFromString (
    "Transform {\n"+
    "  rotation 0 1 0 -1.57\n"+
    "  translation "+x+' '+y+' '+z+'\n'+
    '}'))[0];
    //since PlaneSensors are 2-D, need to rotate z
    //PlaneSensor

  //add zPSensor
  Node[] zPSensor =   //PlaneSensor for z direction
    (Node[])browser.createVrmlFromString (
    "PlaneSensor {\n"+
    "  maxPosition -1 0\n"+
    "  enabled FALSE\n"+
    "}");
  Node[] translator =
    (Node[])browser.createVrmlFromString (
    "Transform { rotation 0 1 0 1.57 }");
    //rotate it back so the rest of the object is turned
    //the right direction

  //add elements to whole
  MFNode adder = (MFNode)whole.getEventIn("addChildren");
  adder.setValue(zPSensor);
  adder.setValue(translator);

  //add route so that the object moves when PlaneSensor
  //senses
//   SFNode zPSensSF = new SFNode (pSensor[0]);
//   SFNode transSF = new SFNode (translator[0]);
  browser.addRoute (zPSensor[0],"translation_changed",
             translator[0],"set_translation");
  dprint ("z pSensor nodes and routes added");

  //add xPSensor
  Node[] xPSensor =   //PlaneSensor for x direction
    (Node[])browser.createVrmlFromString (
    "PlaneSensor {\n"+
    "  maxPosition -1 0\n"+
    "  enabled FALSE\n"+
    "}");
  Node[] otherTranslator =
    (Node[])browser.createVrmlFromString ("Transform { }");

  //add elements to z's translator
  adder = (MFNode)translator[0].getEventIn("addChildren");
  adder.setValue(xPSensor);
  adder.setValue(otherTranslator);

  //add route so that the object moves when PlaneSensor
  //senses
```

```
//   SFNode xPSensSF = new SFNode (pSensor[0]);
//   transSF = new SFNode (otherTranslator[0]);
   browser.addRoute (xPSensor[0],"translation_changed",
            translator[0],"set_translation");
   dprint ("x pSensor nodes and routes added");

   //add yPSensor
   Node[] yPSensor =  //PlaneSensor for y direction
      (Node[])browser.createVrmlFromString (
      "PlaneSensor {\n"+
      "  maxPosition 0 -1\n"+
      "  enabled FALSE\n"+
      "}");
   translator =
      (Node[])browser.createVrmlFromString ("Transform { }");

   //add elements to x's translator
   adder = (MFNode)otherTranslator[0].
           getEventIn("addChildren");
   adder.setValue(yPSensor);
   adder.setValue(translator);

   //add route so that the object moves when PlaneSensor
   //senses
//   SFNode yPSensSF = new SFNode (pSensor[0]);
//   transSF = new SFNode (translator[0]);
   browser.addRoute (yPSensor[0],"translation_changed",
            translator[0],"set_translation");
   dprint ("y pSensor nodes and routes added");

   //add TouchSensor to title bar
   adder = (MFNode)titlebar[0].getEventIn("addChildren");
   adder.setValue (tSensor);

   //adding title bar, object, and ViewPoint to y's translator
   itemHolder = translator;
   adder = (MFNode)translator[0].getEventIn ("addChildren");
   adder.setValue (titlebar);
   adder.setValue (imported);
//   adder.setValue(viewpoint);

   //create ObjInfo, set obj to appropriate value
   Node[] _obj = {whole};     //this roundabout method is
   obj = _obj; //used b/c { } can only be used for array
                //initializations, and not assignments
   return new ObjInfo (titlebar[0],xPSensor[0],
                yPSensor[0],zPSensor[0],x,y,z);
}     //addSensors

 protected Node[] createTitleNode (String title,
        float importWidth, float importHeight, Browser b)
  throws InvalidVRMLSyntaxException
{
  float titleHeight = TITLEHEIGHT;
  float titleWidth = title.length()*TITLEWIDTHRATIO+2*MARGIN;
      //This number is an estimate, because all characters are
      //actually different widths
  float titleY = (importHeight + titleHeight)/(float)2;
      //y position of title bar

  float[] textPos = {-titleWidth+MARGIN,
                -TITLEHEIGHTRATIO/(float)2}; //assumes title is
                   //only one line long
      //textPos = relative to center of title bar, where
      //title's text should be drawn

  if (importWidth>titleWidth) titleWidth = importWidth;
      //so title bar isn't too short

  String titlestr =
  "Billboard {\n"+
  "   bboxSize "+titleWidth+' '+titleHeight+" 0\n"+
  "   children Transform {\n"+
  "     translation 0 "+titleY+" 0\n"+
  "     children [\n"+
  "       Shape {\n"+
  "         appearance Appearance {\n"+
  "           material Material { diffuseColor 1 1 1 }\n"+
  "         }\n"+  //Appearance
  "         geometry Box { size "+titleWidth+' '+titleHeight+" 0 }\n"+
  "       }\n"+ //Shape
  "       LOD {\n"+ //text is only visible within certain range
  "         range "+TITLERANGE+'\n'+
  "         level [\n"+
  "           Transform {\n"+
  "             translation "+textPos[0]+' '+textPos[1]+' '+
                                                    TEXTPROTRUSION+'\n'+
  "             children Shape {\n"+
  "               appearance Appearance {\n"+
  "                 material Material { diffuseColor 0 0 0 }\n"+
  "               }\n"+ //appearance
  "               geometry Text {\n"+   //title bar text
  "                 string \""+title+"\"\n"+
  "                 fontStyle FontStyle {\n"+
  "                   size .25\n"+
  "                   family \"SANS\"\n"+
  "                   style \"BOLD\"\n"+
  "                 }\n"+         //fontStyle
  "               }\n"+  //geometry Text
  "             }\n"+ //Shape
```

```
 "          },\n"+  //Transform
 "          Group { }\n"+
 "        ]\n"+   //level
 "      }\n"+   //LOD
 "    ]\n"+   //children
 "   }\n"+    //Transform (title bar)
 "}";

 return (Node[])b.createVrmlFromString (titlestr);
}

private Node[] createViewPoint (float[] dims, String title,
   Browser b)
throws InvalidVRMLSyntaxException
//requires: dims has 3 elements
//returns a ViewPoint node, facing the object with the given
// dimensions
{
 //calculating distance to stand from object
 //want to fit everything in the screen; so make sure object's
 //not too wide or too tall
 float dist = dims[X]>dims[Y]?dims[X]:dims[Y];

 //make sure we're actually in front of the object, and not
 //inside
 if (dims[Z]/(float)2+VIEWTOOBJ > dist)
   dist = dims[Z]/(float)2+VIEWTOOBJ;

 String str =
 "Viewpoint {\n"+
 "  description \""+viewpointDescription(title)+"\"\n"+
 "  position 0 0 "+dist+'\n'+
 '}';

 dprint ("creating viewpoint from string:\n"+str);
 return (Node[])b.createVrmlFromString (str);
}

//ABSTRACT/PROTECTED METHODS
// public abstract float[] size(); //import's bounding box's
       //dimensions, not including the title bar

 protected abstract Node[] createObj (String title,
              String str, Browser browser, float[] dims)
 throws InvalidVRMLSyntaxException;
   //creates an object. This includes the title bar, but
   //not the touch sensor

// protected abstract BaseNode[] createTitle (String title)
// throws InvalidVRMLSyntaxException;
```

```
    //creates the title bar Transform, position in the correct
    //place relative to the center of the obj.  This Transform
    //will later have the TouchSensor added to it.

  protected abstract String viewpointDescription (String title);
    //what the viewpoint for this import is named in the VRML
    //Browser

  protected void dprint (String s)
  {if (DEBUG) System.out.println (s);}

//PRIVATE METHODS
  private Node[] createScript (String title, Browser b)
  throws InvalidVRMLSyntaxException
//throws InvalidVRMLSyntaxException
  //creates new script node to handle touch events on object
  //used so that object can be moved
  {
    dprint ("creating new TouchTranslator");
    return (Node[])b.createVrmlFromString(
    "Script {\n"+
    "  url \"TouchTranslator.class\"\n"+
//    "  url \"Temp.class\"\n"+   //temporary
    "  field SFString outName \""+title+"\"\n"+
    "  eventIn SFTime touchTime\n"+
    "  eventOut SFString touched\n"+
    '}');
  }   //creatScript

  private Node[] createTSensor (Browser b)
  throws InvalidVRMLSyntaxException
  //creates new touchsensor
  {
    dprint ("creating touch sensor");
    return (Node[])b.createVrmlFromString
                  ("TouchSensor { }");
  }   //createTSensor
} //Import class
```

---

```
/* ImportProto.java
contains ImportProto class.

This class is a superclass of all the objects that can be
added to subdiscussion prototypes.
*/

import vrml.*;
import vrml.node.*;
import vrml.field.*;
import StreamManager;
```

```
public abstract class ImportProto {
//CONSTANTS
public static final boolean DEBUG = Subdiscussion.DEBUG;
public static final int X = 0;
public static final int Y = 1;

//METHODS
public ImportProto (String url, float x, float y,
      float z, Browser b, MFNode addObj)
throws InvalidVRMLSyntaxException
//creates an ImportProto using abstract methods
//adds resulting node to VRML world using addObj
//addObj should correspond to an addChildren eventIn for
//  the subdiscussion prototype
{
  dprint ("creating import prototype: "+url);

  Node[] obj = createObj(url, b);

  //turn up object and move it to the correct position
  obj = transformObj (obj,x,y,z,b);

  //add object to world
  addObj.setValue(obj);
}      //ImportProto constructor

//PROTECTED METHODS
protected abstract Node[] createObj (String url, Browser b)
   throws InvalidVRMLSyntaxException;
   //used to create prototype

protected void dprint (String s)
{if (DEBUG) System.out.println (s);}

//PRIVATE METHODS
private Node[] transformObj (Node[] original, float x, float y,
      float z, Browser b)
throws InvalidVRMLSyntaxException
//turns original up (because subdiscussion prototype's origin
//faces up; so if we turn the object down; the user can see it
//as if it were from the front.  Also translates object using
//x,y,z
{
  Node[] transform = (Node[])b.createVrmlFromString (
  "Transform {\n"+
  "  translation "+x+' '+y+' '+z+'\n'+
  "  rotation 1 0 0 -1.57\n"+   //turn it up
  '}');
```

```
 //put original in transform & return transform
 ((MFNode)transform[0].getEventIn("addChildren")).
   setValue (original);
 return transform;
}      //transformObj
}      //ImportProto
```

---

```
/* ObjInfo.java
contains ObjInfo class

This class is used by ObjMover enable object moving.  It is
mostly a read-only data structure that contains the title bar,
(so the moving axes can be added)  and the 3 PlaneSensors used
to move the object.  The only thing that can be modified
multiple times after an ObjInfo is constructed is the pos.
Dims can be modified after construction through setDims, but
only once.
*/

import vrml.*;
import vrml.node.*;
import vrml.field.*;

public class ObjInfo {
//CONSTANTS
public static final boolean DEBUG = true;
public static final int X = 0;
public static final int Y = 1;
public static final int Z = 2;

//FIELDS
//Node[] obj;     //entire object, with title and sensors
Node axisHolder;    //node to which axes should be added
Node[] pSensors = {null,null,null};
      //"set_enable" field of PlaneSensors
float pos[] = {0,0,0};
float dims[] = {0,0,0};
boolean dimsSet = false;    //when this is true, setDims won't
         //work any more

//METHODS
public ObjInfo (Node a, Node x, Node y, Node z,
         float x2, float y2, float z2)
{
//  obj = o;
 axisHolder = a;
 pSensors[X] = x;
 pSensors[Y] = y;
 pSensors[Z] = z;
 pos[X] = x2;
 pos[Y] = y2;
 pos[Z] = z2;
}     //ObjInfo constructor

//public Node[] getObj() {return obj;}
public Node axisHolder() { return axisHolder;}
public Node xSensor() {return pSensors[X];}
public Node ySensor() {return pSensors[Y];}
public Node zSensor() {return pSensors[Z];}

public float[] getPos()
//returns a copy of pos
{
 return StreamManager.copyDims (pos);
}     //getPos

public void setPos(float[] p)
//requires: p has 3 elements
{pos = StreamManager.copyDims(p);}

public float[] getDims()
//returns a copy of dims
{return StreamManager.copyDims(dims);}

public void setDims (float[] d)
//requires: d has 3 elements
{
 if (DEBUG) System.out.println
     ("setting dims to "+StreamManager.unparseTriple(d));
 if (dimsSet) {    //this should never happen
   System.err.println ("Error: trying to set dimensions in"
     +"ObjInfo after initialization");
   return;
 }
 dimsSet = true;  //so they can't be set again
 dims = StreamManager.copyDims(d);
}     //setDims
} // ObjInfo
```

---

```
/* ObjMover.java
contains class ObjMover

ObjMover is a script that deals with moving objects. When a
title bar is clicked on within the world, ObjMover causes a
set of axes to appear. The user then clicks on one of these
axes to decide the direction in which to move the object.
Then, when the user clicks on the object and drags, it is moved
in the chosen direction. When the user clicks on the title
bar again, the axes disappear, and the change in position is
sent to the server. The server then records the change on
```

```
disk.  It also notifies other users in the discussion of the
movement.
*/

import vrml.*;
import vrml.field.*;
import vrml.node.*;
import java.util.*;
//import TouchTranslator;
import ObjInfo;
import StringCode;
import StreamManager;

public class ObjMover {
//CONSTANTS
  public final static boolean DEBUG = true;
  public final static StringCode sc = new StringCode();
  private final static SFColor HIGHLIGHTCOLOR =
    new SFColor ((float).7,(float).7,(float).7);
       //emissive color that is used to highlight axes
  private final static SFColor NOHIGHLIGHT =
    new SFColor (0,0,0);
  private final static int HIDDEN = -1;  //value for axisMoving
          //when axes are hidden.  Also value used to hide
          //axes
  private final static int SHOWN = 0;  //used to show axes
  private final static int NONE = 4;  //used to determine
  private final static int X = 0;    //along which axis
  private final static int Y = 1;   //everything is being moved
  private final static int Z = 2;  //only used if axes are
                  //visible ( not hidden)
  private static final float[] INITDIMS = {0,0,0}; //default
    //value for dims


//FIELDS
  StreamManager output;

  //used to determine size
  float[] posDims = StreamManager.copyDims(INITDIMS);
  float[] negDims = StreamManager.copyDims(INITDIMS);
  boolean setupChanged = false; //whether objects have been
    //moved, added, or removed since last time the size was
    //calculated

  //received from VRML file through WorldHandler
  SFBool setExaminer;
  SFColor[] changeColor = {null,null,null};
  SFBool[] enablePSensor = {null,null,null};
  Node[] axes;
```

```
      //added to world when user clicks on a title bar
  Node client;   //SFNode in VRML world corresponding to
                 //ConfClient script
  Browser browser;

  //lists of imports
  Hashtable Images = new Hashtable(1);
   //keys are title of Pictures; values are ObjInfos
  Hashtable vrmlObjs = new Hashtable(1);
  Hashtable Comments = new Hashtable (3);
   //keys are speakers & id #; values are ObjInfos

  //used when an object is being moved
  int axisMoving = HIDDEN;
  float[] pos = {0,0,0};
  ObjInfo curObj;  //current object being moved
//  String movedType=null;  //comment or import
  String curTitle;  //title of object being moved

//PUBLIC METHODS
  public ObjMover(SFBool se, SFColor chXCol,
   SFColor chYCol, SFColor chZCol, SFNode a, SFBool enXSen,
   SFBool enYSen, SFBool enZSen, SFNode cl, Browser b,
   StreamManager out)
  {
   try {
    dprint ("constructing ObjMover");

    //getting fields
    setExaminer = se;
    changeColor[X] = chXCol;
    changeColor[Y] = chYCol;
    changeColor[Z] = chZCol;
    enablePSensor[X] = enXSen;
    enablePSensor[Y] = enYSen;
    enablePSensor[Z] = enZSen;
    Node[] a2 = {(Node)a.getValue()};
    axes = a2;
    client = (Node)cl.getValue();
    browser = b;
    output = out;
   } catch (Exception e) {StreamManager.showError(e);}
   dprint ("ObjMover initialization finished");
  }    //ObjMover constructor

  public void reset()
  //resets dimensions of discussion, and import lists
  //if user is moving something, hides Axes, and notifies server
  {
   //check whether anything's being moved
```

```
 if (curObj!=null)
   hideAxes();

 //return to initial state
 setupChanged = false;
 posDims = negDims = StreamManager.copyDims(INITDIMS);
 Comments = new Hashtable(3);
 Images = new Hashtable(1);
 vrmlObjs = new Hashtable(1);

}

public void addComment (String id, ObjInfo info)
//used when comments are added
{
 setupChanged = true;
 Comments.put (id,info);
}    //addComment

public void addImage (String title, ObjInfo info)
//used when an import has been added
//adds given ObjInfo & title to Images
{
 setupChanged = true;
 Images.put (title,info);
}    //addImage

public void add3DObj (String title, ObjInfo info)
//used when an import has been added
//adds given ObjInfo & title to vrmlObjs
{
 setupChanged = true;
 vrmlObjs.put (title,info);
}    //add3DObj

public void removeImport (String type, String id)
{
 setupChanged = true;

 //first, check whether import being removed is also the one
 //being moved
 if (id.equals(curTitle))
   hideAxes();

 //possible bug: what happens if user is dragging as the
 //  object is removed?

 findImportList(type).remove(id);
}
```

```
public void processEvent (Event evt) {
 dprint ("processing event in ObjMover:"+evt.getName());

 try {
 if (evt.getName().equals ("titleClicked")) {
  //title bar was clicked on
  if (axisMoving==HIDDEN) showAxes(((ConstSFString)evt.getValue()).getValue());
  else {
   unclampAxis (axisMoving);  //unhighlight it
   hideAxes();
  }
 }      //if evt = touchTime

 else if (evt.getName().equals ("xClicked")) {
  //X axis was clicked on -- let user move
  clamp(X);
 }

 else if (evt.getName().equals ("yClicked")) {
  //Y axis was clicked on -- let user move
  clamp(Y);
 }

 else if (evt.getName().equals ("zClicked")) {
  //Z axis was clicked on -- let user move
  clamp(Z);
 }

 else if (evt.getName().equals ("translation_changed")) {
  //object was clicked on, and mouse was dragged
  interpretTranslation ((ConstSFVec3f)evt.getValue());
 } //if evt = translation_changed
 } catch (Exception e) {StreamManager.showError(e);}
} //processEvent

public void moveObj (String type, String id, float[] newpos)
//moves object of type given, with given id to newpos
//this is called when a different user moves the object, and
//   the server notifies this world that the object should
//   be moved
//if the local user is also moving the object, this message
//   is ignored. In other words, whoever moves the object
//   last gets the final say
{
 //check whether it's being moved
 if (curTitle.equals(id)) return;

 //Ok to proceed
 setupChanged = true;
 ObjInfo info = (ObjInfo)((Hashtable)findImportList(type)).get(id);
```

```
   info.setPos(newpos);
  }      //moveObj

  public void sendSize()
  //sends size of the world to the server through o
  {
   dprint ("sending size of world");

   //if any objects have been added or moved since last time
   //dims was calculated, then recalculate
   if (setupChanged) calculateDims();

   //send message to server
   output.send (sc.WORLDSIZE,output.unparseTriple(negDims),
           output.unparseTriple(posDims));
  }

  public void shutdown ()
  {dprint ("shutdown called in ObjMover");}

//PRIVATE METHODS
  private void showAxes(String id)
  {
   dprint ("showing axes");

//   axisDisplayer.setValue(SHOWN);

   setupChanged = true;
   curTitle = id;

   //search through all import lists for id
   curObj = findInfo (id);
   if (curObj==null) {
    //something's wrong
    System.err.println
     ("Error: titleClicked could not be processed");
    return;
   }     //if curObj

/*
   curObj = (ObjInfo)Imports.get(id);
   if (curObj == null) {   //a comment is being moved
    movedType = sc.COMMENTSTRING;
    curObj = (ObjInfo)Comments.get(title);
   }     //if curObj
   else movedType = sc.OBJSTRING;  //even if it's actually
      //a picture, it won't matter, cuz the ObjMover can't
      //distinguish between the two
*/
```

```
    axisMoving = NONE;

    //making axes appear
    MFNode adder = (MFNode)curObj.axisHolder().
              getEventIn("addChildren");
    adder.setValue(axes);

    //adding routes to PlaneSensors
    browser.addRoute (client,"enableXPSensor",
              curObj.xSensor(),"set_enable");
    browser.addRoute (client,"enableYPSensor",
              curObj.ySensor(),"set_enable");
    browser.addRoute (client,"enableZPSensor",
              curObj.zSensor(),"set_enable");

    //setting pos
    pos = curObj.getPos();

    //changing navigator to EXAMINE
    setExaminer.setValue(true);
  }

  private void hideAxes()
  //hides axes
  {
    dprint ("hiding axes");
/*  axisDisplayer used to be a Switch node with the axes
    axisDisplayer.setValue(HIDDEN);
*/


    //making axes disappear
    MFNode remover = (MFNode)curObj.axisHolder().
              getEventIn("removeChildren");
    remover.setValue(axes);

    //removing routes from PlaneSensors
    browser.deleteRoute (client,"enableXPSensor",
              curObj.xSensor(),"set_enable");
    browser.deleteRoute (client,"enableYPSensor",
              curObj.ySensor(),"set_enable");
    browser.deleteRoute (client,"enableZPSensor",
              curObj.zSensor(),"set_enable");

    //changing navigator back to FLY
        //(or whatever it was before)
    setExaminer.setValue(false);

    //since object has been moved, setup has been changed
    setupChanged = true;
    //sending message to server
    try {
       output.send (sc.OBJMOVEDSTRING,
          getType(curTitle),curTitle,
          output.unparseTriple(pos));
    } catch (Exception e) {StreamManager.showError(e);}

    //cleaning up
    curObj.setPos (pos);
    curObj = null;
    curTitle = null;
//   movedType = null;
    axisMoving = HIDDEN;
   }

  private void clamp (int clampedDir)
  //requires: clampedDir = X, Y, or Z  (0, 1, 2)
  //       axisMoving = X, Y, Z, or NONE (0,1,2,4)
  //highlights clampedDir axis and lets user move object along
  //it
  {
    dprint ("clamping axis:"+clampedDir);
    if (axisMoving==clampedDir) return;  //no changes needed

    //unhighlight first unclamped axis (if necessary),
    unclampAxis (axisMoving);
    clampAxis (clampedDir);
    axisMoving = clampedDir;

   }    //clamp

  private void interpretTranslation (ConstSFVec3f tr) {
//   dprint ("interpreting translation");
//following line not needed because showAxes takes care of it
//   setupChanged = true;
    if (axisMoving == X)
      pos[X] = tr.getX();
    else if (axisMoving == Y)
      pos[Y] = tr.getY();
    else    //axisMoving == Z
      //because PlaneSensor is 2D, ZSENSOR needed to be
      //rotated, therefore, change in x direction given by
      //tr is actually change in z direction
      pos[Z] = tr.getX();
    dprint ("new position: "+pos[X]+','+pos[Y]+','+pos[Z]);
   }  //interpretTranslation

  private void unclampAxis (int axis)
  //unhighlights axis and disenables its PlaneSensor
```

```
{
 if (axis == NONE) return;   //can't unclamp it
 changeColor[axis].setValue(NOHIGHLIGHT);
 enablePSensor[axis].setValue(false);
}       //unclampAxis

private void clampAxis (int axis)
//highlights axis and enables it PlaneSensor
{
 changeColor[axis].setValue(HIGHLIGHTCOLOR);
 enablePSensor[axis].setValue(true);
}  //clampAxis

private void calculateDims()
//calculates dimensions of world by finding outermost objects
//posDims contains the most positive x,y, & z of the
//edges of the bounding boxes of the outermost objects;
//negDims contains the most negative x,y, & z's
{
  dprint ("calculating dimensions of world");

  //indicate that dims has been updated
  setupChanged = false;

  if (Comments.size()+Images.size()+vrmlObjs.size()==0) {
    //set negDims and posDims to defaults
    negDims = output.copyDims(INITDIMS);
    posDims = output.copyDims(INITDIMS);
    return;
  }       //if Imports.size()

  //there is at least one comment or import; proceed
  ObjInfo nextInfo;   //ObjInfo for next object
  float[] nextPos;    //position of next object
  float[] nextSize;   //size of next object
  Enumeration infos;  //elements of current import list

  //first, check Comments
  if (Comments.size()>0) {
    //set negDims & posDims to low and high bounds of first
    //Import
    infos = Comments.elements();
    nextInfo = (ObjInfo)infos.nextElement();
    negDims = lowBounds(nextInfo);
    posDims = highBounds(nextInfo);

    //traverse through rest of Comments to find outermost objects
    traverseImportDims (infos);
  }       //if Imports.size()
```

```
    //then, check Images
    infos = Images.elements();
    if (Comments.size()==0 && Images.size()>0) {
      //set negDims & posDims to low and high bounds of first
      //Image.
      nextInfo = (ObjInfo)infos.nextElement();
      negDims = lowBounds(nextInfo);
      posDims = highBounds(nextInfo);
    }     //if Imports.size

    //traverse through rest of Images to find outermost objects
    traverseImportDims (infos);

    //then, check vrmlObjs
    infos = vrmlObjs.elements();
    if (Comments.size()+Images.size()==0 && vrmlObjs.size()>0) {
      //set negDims & posDims to low and high bounds of first
      //Image.
      nextInfo = (ObjInfo)infos.nextElement();
      negDims = lowBounds(nextInfo);
      posDims = highBounds(nextInfo);
    }     //if Imports.size

    //traverse through rest of Images to find outermost objects
    traverseImportDims (infos);

    dprint ("dimensions calculated:"+
        StreamManager.unparseTriple(negDims)+' '+
        StreamManager.unparseTriple(posDims));
  }    //calculateDims

  //LOWER-LEVEL PRIVATE FUNCTIONS
private ObjInfo findInfo (String id)
//traverses through all import lists and returns the
//corresponding ObjInfo for id
//if id is not found, returns null
{
  ObjInfo returnval = (ObjInfo)Comments.get(id);
  if (returnval != null) return returnval;
  if ((returnval=(ObjInfo)Images.get(id))!=null)
    return returnval;
  if ((returnval=(ObjInfo)vrmlObjs.get(id))!=null)
    return returnval;

  //if we've gotten this far, id was not found
  System.err.println ("Error: ObjInfo not found in ObjMover:"+id);
  return null;
}     //findInfo

private String getType (String id)
```

```
//finds which list id is in
//String returned is COMMENTSTRING, PICWINSTRING, OBJSTRING.
//if it's in none of the lists, returns null
{
 if (Comments.containsKey(id))
   return sc.COMMENTSTRING;
 if (Images.containsKey(id))
   return sc.PICWINSTRING;
 if (vrmlObjs.containsKey(id))
   return sc.OBJSTRING;
 return null;
}      //getType

private Hashtable findImportList (String list)
//returns Comments, Images, or vrmlObjs
//if list doesn't match an appropriate string, returns null
{
 if (list.equals(sc.COMMENTSTRING)) return Comments;
 if (list.equals(sc.PICWINSTRING)) return Images;
 if (list.equals(sc.OBJSTRING)) return vrmlObjs;
 else System.err.println ("Error: Import list type not found:"
      +list);
 return null;
}

 private float[] lowBounds (ObjInfo info)
 //returns the corner of the bounding box of object
 //corresponding to info with the lowest X,Y,and Z coordinates
 {
   //get position and size
   float[] objPos = info.getPos();
   float[] objSize = info.getDims();
   dprint ("pos of object:"+StreamManager.unparseTriple(objPos));
   dprint ("size of object:"+StreamManager.unparseTriple(objSize));

   //find appropriate corner.  All size values are divided by
   //two because pos is measured from the center of the object
   float[] returnval = {
      objPos[X]-objSize[X]/(float)2,
      objPos[Y]-objSize[Y]/(float)2,
      objPos[Z]-objSize[Z]/(float)2
   };
   return returnval;
 }    //lowBounds

 private float[] highBounds (ObjInfo info)
 //returns the corner of the bounding box of object
 //corresponding to info with the greatest X,Y,and Z coordinates
 {
   //get position and size
```

```
    float[] objPos = info.getPos();
    float[] objSize = info.getDims();

    //find appropriate corner.  All size values are divided by
    //two because pos is measured from the center of the object
    float[] returnval = {
      objPos[X]+objSize[X]/(float)2,
      objPos[Y]+objSize[Y]/(float)2,
      objPos[Z]+objSize[Z]/(float)2
    };
    return returnval;
  }   //lowBounds

private void traverseImportDims (Enumeration infos)
//requires: infos is an Enumeration of ObjInfos
//modifies:  this.negDims, this.posDims
//traverses through infos and finds out whether each object
//corresponding to the ObjInfo is outside the boundaries
//designated by negDims and posDims.  If not, it modifies
//negDims and/or posDims appropriately.
{
  float[] nextLBounds;   //low bounds of next object
  float[] nextHBounds;  //high bounds
  ObjInfo nextInfo;

  while (infos.hasMoreElements()) {
    //find out stuff about next object
    nextInfo = (ObjInfo)infos.nextElement();
    nextLBounds = lowBounds(nextInfo);
    nextHBounds = highBounds(nextInfo);

    //if any of the bounds are more extreme than negDims
    //or posDims, update appropriate dims
    if (nextLBounds[X] < negDims[X]) negDims[X]=nextLBounds[X];
    if (nextLBounds[Y] < negDims[Y]) negDims[Y]=nextLBounds[Y];
    if (nextLBounds[Z] < negDims[Z]) negDims[Z]=nextLBounds[Z];
    if (nextHBounds[X] > posDims[X]) posDims[X]=nextHBounds[X];
    if (nextHBounds[Y] > posDims[Y]) posDims[Y]=nextHBounds[Y];
    if (nextHBounds[Z] > posDims[Z]) posDims[Z]=nextHBounds[Z];
  }   //while infos.hasMoreElements
}     //traverseImportDims

  private void dprint (String s)
  {if (DEBUG) System.out.println (s);}
}     //ObjMover
```

---

```
/* Picture.java
includes Picture class

This class is a representation of imported 2-d images into the VRML world.
```

```
*/

import vrml.*;
import vrml.field.*;
import vrml.node.*;
import java.awt.*;
import Import;

public class Picture extends Import {
//CONSTANTS
private static final float PICRATIO = (float).02;
      //ratio to multiply by when
      //converting image size from downloading for applet to image size
      //when placed in VRML world

//FIELDS
//float imageWidth;      //dimensions of image
//float imageHeight;

//CONSTRUCTORS
  public Picture (String title, String url, float x, float y, float z,
          Browser b, MFNode addObj, SFNode cl)
  throws InvalidVRMLSyntaxException, InterruptedException
  {super(title,url,x,y,z,b,addObj,cl);}


//METHODS
protected Node[] createObj (String title, String url,
      Browser browser, float[] dims)
  throws InvalidVRMLSyntaxException
  //creates picture with given title and url
{
  dprint ("creating Image in world");

  float[] imageSize = getImageSize(url);
  dims[X] = dims[Z] = imageSize[X];
  dims[Y] = imageSize[Y];

  String imagestr =
  "Billboard {\n"+
  "  bboxSize "+dims[X]+' '+dims[Y]+" 0\n"+
  "  children Shape {\n"+
  "    appearance Appearance {\n"+
  "      texture ImageTexture { url "+url+" }\n"+
  "    }\n" +
  "    geometry Box { size "+dims[X]+' '+dims[Y]+" 0 }\n"+
  "  }\n"+  //Shape
  '}';

  //create the actual nodes
  dprint ("creating image from string:"+imagestr);
  Node [] imagenode =
    (Node[])browser.createVrmlFromString (imagestr);
  Node[] titlenode = createTitleNode(title, dims[X],
                            dims[Y], browser);
  Node[] returnnode = {imagenode[0],titlenode[0]};
  return returnnode;
} //createObj

protected String viewpointDescription (String title)
//description of viewpoint for this object in the VRML Browser
{return "Image: "+title;}

public static float[] getImageSize (String url)
//figures out size of image with given url
{
/*  DOESN'T WORK
  //calculating image size
  Canvas loader = new Canvas();     //needed as an ImageLoader for picture
  MediaTracker tracker = new MediaTracker (loader);

  //need to download to find out image size
  dprint ("downloading picture to determine image size");
  Image pic = loader.getToolkit().getImage(url);
  tracker.addImage(pic, 0);
  try {        //this try/catch block is needed b/c the compiler complains
        //that InterruptedException must be declared in throws clause,
        //but when it is declared, it complains that
        //InterruptedException is not allowed to be thrown
    tracker.waitForID(0);
  } catch (Exception e) {
    System.err.println ("Exception thrown: ");
    e.printStackTrace();
    return null;
  }

  dprint ("Image size before scaling: "+pic.getWidth(loader)+','
        +pic.getHeight(loader));

  //scale for vrml world
  dims[X] = dims[Z] = pic.getWidth(loader)*PICRATIO;
  dims[Y] = pic.getHeight(loader)*PICRATIO;
  dprint ("Image size calculated: "+imageWidth+','+imageHeight);
*/
   //should be removed once size of images can be figured
  float[] returnval = {3,3};
  return returnval;
}      //getImageSize
}     //Picture
```

```
/*PictureProto.java
contains PictureProto class.

This class is the prototype used for images that are added
to subdiscussion prototypes. PictureProtos are just like
Pictures, except that they have no title bar, are not Billboards,
and can not be moved. Also, when they are added to a
Subdiscussion prototype, they are scaled to fit the designated
room for the subdiscussion.
*/

import vrml.*;
import vrml.node.*;
import vrml.field.*;
import ImportProto;

public class PictureProto extends ImportProto {
//METHODS
public PictureProto (String url, float x, float y, float z,
    Browser b, MFNode addObj)
throws InvalidVRMLSyntaxException
//x,y,z = position at which to place PictureProto
//addObj = eventIn used to add PictureProto. This eventIn
//   should be the eventIn of the Subdiscussion prototype
{super (url,x,y,z,b,addObj);}

protected Node[] createObj (String url, Browser b)
   throws InvalidVRMLSyntaxException
//url = url of Image to be imported
{
  float[] dims = Picture.getImageSize(url);

  return (Node[])b.createVrmlFromString (
  "Shape {\n"+
  "  appearance Appearance {\n"+
  "    texture ImageTexture { url "+url+" }\n"+
  "  }\n" +
  "  geometry Box { size "+dims[X]+' '+dims[Y]+" 0 }\n"+
  "}\n"+ //Shape
  '}');
} //createObj
}     //PictureProto
```

```
/* Subdiscussion.java
includes Subdiscussion class

This class is a the representation of subidiscussion prototypes.
In the VRML world, subdiscussions are represented as rooms with
semi-transparent walls. They are all lined up next to each
```

```
other. When the user clicks on the door to the room, he is
enterred into that subdiscussion. When the suer views the room,
he sees a bird's eye view of the discussion, except that the
objects are turned so that they are facing him. In other words,
the user can see the layout of the room, and the objects are
all turned toward him.
*/

import vrml.*;
import vrml.field.*;
import vrml.node.*;
import java.awt.Dimension;
import CommentProto;
import PictureProto;
import ThreeDProto;
import StringCode;

public class Subdiscussion {
//CONSTANTS
public final static boolean DEBUG = WorldHandler.DEBUG;
public final static StringCode sc = new StringCode();
public final static int X = 0;
public final static int Y = 1;
public final static int Z = 2;


private final static double[] DOORCOLOR = {.9,.9,.9}; //almost white
private final static float DOORWIDTH = 4;
private final static float DOORHEIGHT = 7;
private final static double[] WALLCOLOR = {.1,.1,.7}; //light blue
private final static float WALLTRANSPARENCY = (float).6; // the
   //transparency of the walls of the prototype
private final static float WALLMARGIN = (float).5;  //minimum distance
   //between a middle of wall and the nearest object. In other
   //words, half the thickness of the wall is included in this
   //number
private final static float WALLTHICKNESS = (float).5; //thickness
                    //of walls
private final static float STANDARDHEIGHT = (float)7.5;
      //the standard height of most subdiscussion rooms.
      //if, when scaled, the subdiscussion still fits the
      //constraints below, the STANDARDHEIGHT is used. Otherwise,
      //the room size is modified appropriately

//the following 3 constants are for determining how to scale the
//subdiscussion. MINSIZE refers to the minimum values for the
//width, height, and depth allocated for the room.
//MINSCALEDDIMS and MAXSCALEDDIMS refer to the minimum and maximum
//values of the positions of the outermost objects. Note that
//the space allocated for the room (which is affected by MINSIZE
//determines the size of the walls of the subdiscussion prototype,
//while the positions of the outermost objects are subject to
//MAXSCALEDDIMS and MINSCALEDDIMS. In other words, the value
//by which to scale the subdiscussion's layout (to shrink or
//enlarge it) is first calculated using MAXSCALEDDIMS and
//MINSCALEDDIMS. Then, if any of the dimensions are smaller than
//those specified by MINSIZE, extra space is added, even though
//the outermost objects may be more than WALLMARGIN away from
//the walls.
//Because it is desired to preserve the ratio of width to height
//to depth of the subdiscussion (i.e., it is scaled by the same
//number for all three dimensions), it may not always be possible
//to satisfy both the MAXSCALEDDIMS and the MINSCALEDDIMS at the
//same time. (more specifically, for the narrow discussions) In
//this case, the MAXSCALEDDIMS overrides the MINSCALEDDIMS.
//For more information, see the calculateScale() procedure.

private final static float[] MINSIZE = {DOORWIDTH,DOORHEIGHT,3};
private final static Dimension MAXSCALEDDIMS = new Dimension (20,20);
private final static Dimension MINSCALEDDIMS = new Dimension (3,3);
private final static float MAXVIEWTOROOMDISTANCE = 8;
   //maximum distance that the Viewpoint can be to a subdiscussion
   //room. If the Viewpoint is too far, imports of the current
   //discussion room may get in the way.

//FIELDS
Node[] room;        //where all objects are put
String title;       //of discussion
Node[] tSensor;     //TouchSensor for door
Node[] script;      //TouchTranslator
//Node[] view;       //Viewpoint
Node client;
MFNode addToRoom;  //event that lets objects be added to the
                   //room
float[] dims = {0,0,0};        //dimensions of prototype
Browser browser;

//PUBLIC METHODS
public Subdiscussion(String t, MFNode addDisc, SFNode c, Browser b,
   float pos, float[] negDims, float[] posDims)
throws InvalidVRMLSyntaxException
//requires: negDims & posDims each have 3 entries
//pos = position of lower right corner of Subdisc prototype's
// the first room will have pos = 0; the second will have pos
// -(width of first room); the third will have pos =
// -(width of first + width of second); etc. In other words,
// rooms will be lined up adjacent to each other, and each
// subsequent room will be to the left of the current one.
//negDims = smallest/most negative coordinates that any object
// occupies in this subdiscussion. Note that these coordinates
// refer to the ones when the subdiscussion is NOT scaled (i.e.,
// the size it would be if the user were t enter this
// subidscussion.)
//posDims = same, but most positive coordinates instead
{
  dprint ("creating subdiscussion");

  title = t;
  client = (Node)c.getValue();
  browser = b;

  //calculating dimensions of room
  float ratio = calculateScale(negDims,posDims);
   //= ratio of dimensions of room prototype to dimensions of
   // actual discussion.
  dims[X] = (posDims[X]-negDims[X])*ratio
            + WALLMARGIN*(float)2; //width of room
  dims[Y] = (posDims[Y]-negDims[Y])*ratio //height
            + WALLMARGIN*(float)2;
  dims[Z] = (posDims[Z]-negDims[Z])*ratio + WALLMARGIN; //depth
      //don't need to multiply WALLMARGIN by two because there
      //is no back wall

  //calculating offset
  //offset = where to put the origin of the subdiscussion,
  //   relative to position (0,0,0). See below for more
  //   information
  float[] offset = calculateOffset (negDims,posDims,ratio);

  //make sure room fits MINSIZE restriction
  fitToSize (dims,offset);

  //creating room Node
  String roomstr =
    "Transform {\n"+
    "  translation "+pos+" 0 0\n"+
    "}";
  room = (Node[])b.createVrmlFromString (roomstr);
  MFNode addWall = (MFNode)room[0].getEventIn ("addChildren");

  //adding door
  addWall.setValue (createDoor());

  //right wall
  float[] wallPos = {0,dims[Y]/(float)2,-dims[Z]/(float)2};
          //position of wall
  float[] wallRot = {0,1,0,(float)1.57}; //rotation of wall
  addWall.setValue
      (createWall(dims[Z],dims[Y],wallPos,wallRot,0));
  //left wall
  wallPos[X] = -dims[X];
  addWall.setValue
      (createWall(dims[Z],dims[Y],wallPos,wallRot,0));

  //front wall
  wallPos[X] = -dims[X]/(float)2;
  wallPos[Z] = 0;
  wallRot[3] = 0;      //no rotation
  addWall.setValue
      (createWall(dims[X],dims[Y],wallPos,wallRot,
              WALLTRANSPARENCY));
  //viewpoint
  //no longer gets added, because removal doesn't work
//  addWall.setValue (view=createViewPoint (t,dims));

  //adding scaling Transform to which imports are added
  String objholderstr =
  "Transform {\n"+
  "  translation "+offset[X]+' '+offset[Y]+' '+offset[Z]+'\n'+
  "  rotation 1 0 0 1.57\n"+
  "  scale "+ratio+' '+ratio+' '+ratio+'\n'+
  '}';
  dprint ("creating scaling transform:\n"+objholderstr);
  Node[] objHolder = (Node[])b.createVrmlFromString(objholderstr);
  addWall.setValue (objHolder);
  addToRoom = (MFNode)objHolder[0].getEventIn("addChildren");

  //creating TouchTranslator and adding it
/* doesn't work - trouble reading .class file
  script = (Node[])b.createVrmlFromString(
  "Script {\n"+
  "  url \"TouchTranslator.class\"\n"+
  "  field SFString outName \""+title+"\"\n"+
  "  eventIn SFTime touchTime\n"+
  "  eventOut SFString touched\n"+
  '}');
  dprint ("touchTranslator created for superdiscussion");
  addObj.setValue (script);
*/

   //adding room
   addDisc.setValue(room);

   //adding routes
//   dprint ("adding routes for subdisc door");
//  b.addRoute (tSensor[0],"touchTime",script[0],"touchTime");
//  b.addRoute (script[0],"touched",client,"doorClicked");
}
```

```
public float getWidth() {return dims[X];}

public boolean addProto (String importType, String url,
        float[] coords)
//requires: coords has 3 values
//adds prototype of given importType (comment/picture/etc.)
//at given coordinates (relative to origin of Subdiscussion).
//url is the url for pictures & 3D objects; url is ignored
//  for comments
{
  dprint ("adding prototype to discussion:"+importType+','+url);

  try {
    //note that the constructors automatically add the
    //object to the world
    if (importType.equals(sc.COMMENTSTRING)) {
      new CommentProto (coords[X],coords[Y],coords[Z],browser,
              addToRoom);
      return true;
    }
    if (importType.equals(sc.PICWINSTRING)) {
      new PictureProto (url,coords[X],coords[Y],coords[Z],
          browser,addToRoom);
      return true;
    }
    else if (importType.equals(sc.OBJSTRING)) {
      new ThreeDProto (url,coords[X],coords[Y],coords[Z],
          browser,addToRoom);
      return true;
    }
    else {
      System.err.println
          ("Error: unrecognized object type:"+importType);
      return false;
    }
  } catch (InvalidVRMLSyntaxException e) {
    System.err.println
      ("Error:Prototype not created:"+importType+','+url);
  }    //catch InvalidVRMLSyntaxException
  catch (Exception e) {
    StreamManager.showError(e);
  }    //try/catch
  return false;
}

public void remove (MFNode removeObj)
//uses EventIn removeObj to remove object (w/ touchSensor) & removeScript
//to remove script from world.  Also removes appropriate routes
{
  dprint ("SuperDisc.remove() called");
```

```
// b.deleteRoute
//     (tSensor[0],"touchTime",script[0],"touchTime");
// b.deleteRoute
//     (script[0],"touched",client,"titleClicked");
// removeObj.setValue(script);

  //remove viewpoint
  //no longer gets added -- removal doesn't do anything
// MFNode removeWall = (MFNode)room[0].getEventIn ("removeChildren");
// removeWall.setValue (view);

  //remove room from world
  removeObj.setValue(room);
} //remove

//PRIVATE METHODS
//Methods that create VRML objects
private Node[] createDoor ()
throws InvalidVRMLSyntaxException
//creates a door, with TouchSensors, translated from the lower
//right corner of the subdiscussion
{
  String doorstr =
  "Transform {\n"+
  "  translation -"+dims[X]/2.0+' '+DOORHEIGHT/2.0+" 0\n"+
  "  children [\n"+
//title of discussion should be drawn on the door
  "    Shape {\n"+     //the actual door
  "      appearance Appearance {\n"+
  "        material Material {\n"+
  "          diffuseColor .9 .9 .9\n"+
  "          transparency "+WALLTRANSPARENCY+'\n'+
  "        }\n"+
  "      }\n"+ //appearance
  "      geometry Box { size "+DOORWIDTH+' '+DOORHEIGHT+' '+
              (WALLTHICKNESS+.1)+" }\n"+
        //the "+.1" is so that the door is slightly thicker
        //than the walls; therefore, the door will always
        //be visible because its front is in front of the
        //wall
  "    }\n"+ //Shape
  "    Transform {\n"+     //a doorknob
  "      translation "+(DOORWIDTH/2.0-.7)+" 0 " +
          (WALLTHICKNESS/2.0+.15)+'\n'+
     //place the doorknob so that it's on the right side,
     //in middle (in the up/down direction), and in front of
     //the door
  "      children Shape {\n"+
  "        appearance Appearance {\n"+
 "        material Material {\n"+
 "          diffuseColor "+DOORCOLOR[0]+' '+DOORCOLOR[1]+
                          ' '+DOORCOLOR[2]+'\n'+
 "          transparency "+WALLTRANSPARENCY+'\n'+
 "          shininess .6\n"+
 "        }\n"+
 "      }\n"+ //appearance
 "      geometry Sphere { radius .2 }\n"+
 "    }\n"+ //Shape
 "  }\n"+ //Transform
 " ]\n"+ //children
 "}";

 dprint ("creating subdisc door from string:\n"+doorstr);
 Node[] door =
   (Node[])browser.createVrmlFromString (doorstr);
 dprint ("door created");

 //create TouchSensor and adding it
 tSensor =
   (Node[])browser.createVrmlFromString ("TouchSensor { }");
 MFNode adder = (MFNode)door[0].getEventIn ("addChildren");
 adder.setValue(tSensor);
 dprint ("touchSensor created for subdiscussion");
 return door;
}     //createDoor

private Node[] createWall (float w, float h, float[] trans,
              float[] rot, float transparency)
throws InvalidVRMLSyntaxException
//w,h = width & height of wall
//trans,rot = translation & rotation in Transform node
//transparency = transparency of the wall
{
 String wallstr =
 "Transform {\n"+
 "  translation "+trans[X]+' '+trans[Y]+' '+trans[Z]+'\n'+
 "  rotation "+rot[X]+' '+rot[Y]+' '+rot[Z]+' '+rot[3]+'\n'+
 "  children Shape {\n"+
 "    appearance Appearance {\n"+
 "      material Material {\n"+
 "        diffuseColor "+WALLCOLOR[0]+' '+WALLCOLOR[1]+' '
                       +WALLCOLOR[2]+'\n'+
 "        transparency "+transparency+'\n'+
 "      }\n"+ //material
 "    }\n"+ //appearance
 "    geometry Box {\n"+
 "      size "+w+' '+h+' '+WALLTHICKNESS+'\n'+
 "    }\n"+
 "  }\n"+ //Shape
```

```
  '}';

  dprint ("creating wall from string:\n"+wallstr);
  return (Node[])browser.createVrmlFromString (wallstr);
} //createWall

private Node[] createViewPoint (String title, float[] dims)
//creates viewpoint in VRML browser for this subdiscussion
throws InvalidVRMLSyntaxException
{
  //first, try to fit entire subdiscussion into view
  float dist = dims[X]>dims[Y]?dims[X]:dims[Y];

  //make sure we're not too far away (if the discussion's big)
  //if we're too far, there may be imports in the current discussion
  //room that are in the way
  if (dist>MAXVIEWTOROOMDISTANCE)
    dist = MAXVIEWTOROOMDISTANCE;

  String str =
  "Viewpoint {\n"+
  "  description \"Discussion: "+title+"\"\n"+
  "  position -"+(dims[X]/(float)2)+' '+(dims[Y]/(float)2)+
                 ' '+dist+'\n'+
  '}';

  dprint ("creating viewpoint from string:\n"+str);
  return (Node[])browser.createVrmlFromString (str);
}         //createViewPoint

//Methods to make calculations
private float calculateScale (float[] negDims, float[] posDims)
//requires: negDims & posDims & dims each have 3 entries
//effects: returns ratio of dimensions of room prototype to
//  dimensions of actual subdiscussion.Because the room is turned
//  so that the user will have a bird's eye view, the height of
//  the room corresponds to the depth of the actual
//  discussion, and the depth of the room corresponds to
//  the height
//The value returned is only subject to MAXSCALEDDIMS and
//MINSCALEDDIMS, not to MINSIZE
{
  dprint ("calculating scale for subdisc:"+
      StreamManager.unparseTriple(negDims)+','+
      StreamManager.unparseTriple(posDims));
  float[] unscaledSize =
   {posDims[X]-negDims[X],
   posDims[Y]-negDims[Y],
   posDims[Z]-negDims[Z]};
```

```
 if (unscaledSize[X]==0) return 1;

 //remember that room is turned; so posDims/negDims[Z]
 //corresponds to height of room
 float ratio = STANDARDHEIGHT/(unscaledSize[Z]);

 //checking whether it's too wide
 float width = ratio*unscaledSize[X];
 if (width > MAXSCALEDDIMS.width) {
  //need to squeeze it in X direction
  ratio = MAXSCALEDDIMS.width/unscaledSize[X];
  //since MAXSCALEDDIMS overrides MINSCALEDDIMS, don't need
  //to check whether scaled height is smaller than MINSCALEDDIMS;
  //Also, we know it's smaller than MAXSCALEDDIMS because it's
  //smaller than STANDARDHEIGHT now
  dprint ("room was too wide: ratio = "+ratio);
  return ratio;
 }    //if width

 //checking whether it's too narrow (width-wise)
 if (width > MINSCALEDDIMS.width) {
  //everything's OK
  dprint ("room was just right proportions: ratio = "+ratio);
  return ratio;
 }    //if width

 //too narrow; resize it
 ratio = MAXSCALEDDIMS.width/unscaledSize[X];
 dprint ("room is too narrow; new ratio = "+ratio);

 //check whether resize caused height to go over limit
 float height = ratio*unscaledSize [Z];
 if (height>MAXSCALEDDIMS.height) {
  //since MAXSCALEDDIMS overrides MINSCALEDDIMS, reset ratio,
  //according to height
  ratio = MAXSCALEDDIMS.height/unscaledSize[Y];
  dprint ("room was too tall after resizing: ratio ="+ratio);
 }

 return ratio;
}     //calculateScale

private float[] calculateOffset (float[] negDims, float[] posDims,
          float ratio)
//requires: negDims & posDims & dims each have 3 entries
//effects: returns offset of origin of subdiscussion prototype,
// relative to the lower front corner of the prototype. All
// objects added to the room will be measured from this origin.
{
 float[] offset = {0,0,0};    //allocate space for 3 floats
 offset[X] = -posDims[X]*ratio-WALLMARGIN;
        //posDims negated because we
        //want to move left of pos (because pos indicates the
        //right wall of this subdiscussion prototype)
 offset[Y] = posDims[Z]*ratio+WALLMARGIN; //not negated because we want to move
     //it up.  Takes the Z coordinate (instead of Y) because
     //of the rotation in order to give the user a bird's eye
     //view  (see below)
 offset[Z] = -posDims[Y]*ratio+WALLMARGIN;
     //negated so that we move it out;
     //take posDims[Y] instead of posDims[Z] because of
     //rotation
 dprint ("offset calculated: "+StreamManager.unparseTriple(offset));
 return offset;
}

private void fitToSize (float[] dims, float[] offset)
//requires: dims & offset each have 3 elements
//modifies: dims, offset
//
{
 dprint ("initial value of dims: "+StreamManager.unparseTriple(dims));
 //if any of the dimensions are smaller than those specified
 //by MINSIZE, we want to center the contents of the
 //discussion and change the value of dims
 if (dims[X]<MINSIZE[X]) {
   offset[X] -= (MINSIZE[X]-dims[X])/(float)2; //center it
   dims[X] = MINSIZE[X];
  }   //dims
 if (dims[Y]<MINSIZE[Y]) {
   offset[Y] += (MINSIZE[Y]-dims[Y])/(float)2; //center it
   dims[Y] = MINSIZE[Y];
  }   //dims
 if (dims[Z]<MINSIZE[Z]) {
   offset[Z] = MINSIZE[Z]; //push it back
   dims[Z] = MINSIZE[Z];
  }   //if dims

 dprint ("final value of dims: "+StreamManager.unparseTriple(dims));
 dprint ("final value of offset: "+StreamManager.unparseTriple(offset));
}

//Other private methods
private void dprint (String str)
{if (DEBUG) System.out.println (str);}
}
```

```
/*SuperDisc.java
includes SuperDisc class
```

```
Used to represent superdiscussions.  Contains a Node[] that
represents the door in the VRML world that leads to this
superdiscussion.  Also contains the title of the discussion
and a TouchSensor corresponding to the door.
When the user clicks on the door, it takes him to the
superdiscussion behind the door.
*/

import vrml.*;
import vrml.node.*;
import vrml.field.*;

public class SuperDisc {
//CONSTANTS
public final static boolean DEBUG = WorldHandler.DEBUG;

//FIELDS
Node[] door;
String title;       //of discussion
Node[] tSensor;     //TouchSensor for door
Node[] script;      //TouchTranslator
Node client;

//METHODS
public SuperDisc (String t, MFNode addObj, SFNode c, Browser b)
throws InvalidVRMLSyntaxException
{
 dprint ("creating superdiscussion");

 title = t;
 client = (Node)c.getValue();

 //creating a door on the back wall
 String doorstr =
 "Transform {\n"+
 "  translation 0 -1.5 4.75\n"+
 "  rotation -0 1 0 3.142\n"+
 "  children [\n"+
//title of discussion should be drawn on the door
 "    Shape {\n"+     //the actual door
 "      appearance Appearance {\n"+
 "        material Material { diffuseColor .9 .9 .9 }\n"+
 "      }\n"+ //appearance
 "      geometry Box { size 4 7 .6 }\n"+
 "    }\n"+ //Shape
 "    Transform {\n"+     //a doorknob
 "      translation 1.3 0 .4\n"+
 "      children Shape {\n"+
 "        appearance Appearance {\n"+
 "          material Material {\n"+
"          diffuseColor .8 .8 .2\n"+
"          shininess .6\n"+
"        }\n"+    //material
"      }\n"+ //appearance
"      geometry Sphere { radius .2 }\n"+
"    }\n"+ //Shape
"  }\n"+ //Transform
" ]\n"+ //children
"}";

 dprint ("creating superdisc door from string:\n"+doorstr);
 door =
   (Node[])b.createVrmlFromString (doorstr);
 dprint ("door created");

 //create TouchSensor and adding it
 tSensor =
   (Node[])b.createVrmlFromString ("TouchSensor { }");
 MFNode adder = (MFNode)door[0].getEventIn ("addChildren");
 adder.setValue(tSensor);
 dprint ("touchSensor created for superdiscussion");

 //creating TouchTranslator and adding it
/* doesn't work - trouble reading VRML file
 script = (Node[])b.createVrmlFromString(
 "Script {\n"+
 "  url \"TouchTranslator.class\"\n"+
 "  field SFString outName \""+title+"\"\n"+
 "  eventIn SFTime touchTime\n"+
 "  eventOut SFString touched\n"+
 '}');
 dprint ("touchTranslator created for superdiscussion");
 addObj.setValue (script);
*/

   //adding door and routes
 addObj.setValue (door);
// b.addRoute (tSensor[0],"touchTime",script[0],"touchTime");
// b.addRoute (script[0],"touched",client,"doorClicked");

// the following line only used in testing phase
 b.addRoute (tSensor[0],"touchTime",client,"doorClicked");
}      //SuperDisc constructor

public String title() {return title;}

 public void remove (MFNode removeObj, Browser b)
 //uses EventIn removeObj to remove object (w/ touchSensor) & removeScript
 //to remove script from world.  Also removes appropriate routes
 {
    dprint ("SuperDisc.remove() called");

    //the following line only used in testing phase

    b.deleteRoute (tSensor[0],"touchTime",client,"doorClicked");
//  b.deleteRoute
//     (tSensor[0],"touchTime",script[0],"touchTime");
//  b.deleteRoute
//     (script[0],"touched",client,"titleClicked");
//  removeObj.setValue(script);
    removeObj.setValue(door);

  } //remove

private void dprint (String str)
{if (DEBUG) System.out.println (str);}
}
```

---

```
/* ThreeDObj.java
contains ThreeDObj class

Used to represent imported 3-d objects in Vrml world.
*/

import vrml.*;
import vrml.field.*;
import vrml.node.*;
import Import;

public class ThreeDObj extends Import {
//CONSTANTS
private static final double PICRATIO = .02; //ratio to multiply by when
     //converting image size from downloading for applet to image size
     //when placed in VRML world

//FIELDS
//float objWidth;
//float objHeight;
//float objDepth;

//CONSTRUCTORS
  public ThreeDObj (String title, String url, float x, float y,
      float z, Browser b, MFNode addObj, SFNode cl)
  throws InvalidVRMLSyntaxException
  {super(title,url,x,y,z,b,addObj,cl);}

//METHODS
protected Node[] createObj (String title, String url,
      Browser browser, float[] dims)
  throws InvalidVRMLSyntaxException
```

```
 //creates picture with given title and url
{
 dprint ("creating 3-D object in world");

 //calculating object's size - NEEDS TO BE CHANGED
// float objWidth = 3;
// float objHeight = 5;
// float objDepth = 3;
 dims[X] = dims[Z] = 3;
 dims[Y] = 5;

 String objstr = "Inline { url \""+url+"\" }";

 //create the actual nodes
 dprint ("creating 3-D object from string:\n"+objstr);
 Node [] objnode =
  (Node[])browser.createVrmlFromString (objstr);
 Node[] titlenode = createTitleNode(title, dims[X],
                        dims[Y], browser);
 Node[] returnnode = {objnode[0],titlenode[0]};
 return returnnode;
} //createObj


protected String viewpointDescription (String title)
//description of viewpoint for this object in the VRML Browser
{return "VRML: "+title;}
}
```

---

```
/*ThreeDProto.java
contains ThreeDProto class.

This class is the prototype used for 3-D objects that are added
to subdiscussion prototypes. ThreeDProtos appear just like
ThreeDObjs, but without a title bar. Also, when they are added
to the subdiscussion prototype, they are scaled to fit the
designated room.
*/

import vrml.*;
import vrml.node.*;
import vrml.field.*;
import ImportProto;

public class ThreeDProto extends ImportProto {
//METHODS
public ThreeDProto (String url, float x, float y, float z,
    Browser b, MFNode addObj)
throws InvalidVRMLSyntaxException
//x,y,z = position at which to place ThreeDProto
//addObj = eventIn used to add ThreeDProto.  This eventIn
//   should be the eventIn of the Subdiscussion prototype
{super (url,x,y,z,b,addObj);}

protected Node[] createObj (String url, Browser b)
   throws InvalidVRMLSyntaxException
//url = url of Image to be imported
{
  dprint ("create 3-D prototype: "+url);
  return (Node[])b.createVrmlFromString
    ("Inline { url \""+url+"\" }");
}         //createObj
}     //ThreeDProto
```

---

```
/* TouchTranslator.java
contains TouchTranslator class.

This class simply takes in an eventIn and creates an eventOut
under a different name.  It is needed because scripts can not
distinguish between events of the same name coming from
different sources.  For instance, let's say we have the following
routes in the VRML file:

ROUTE TOUCHSENSOR1.touchTime TO SCRIPT.eventIn1
ROUTE TOUCHSENSOR2.touchTime TO SCRIPT.eventIn2

Then in processEvent(Event evt), we call evt.getName().  No
matter whether the user clicked on the object corresponding to
TOUCHSENSOR1 or TOUCHSENSOR2, evt.getName() would yield the
same result: "touchTime"  Perhaps this problem is merely a bug
in Cosmo Player Beta v2.0 for Windows 95, or perhaps this
behavior is intended.
*/

import vrml.*;
import vrml.field.*;
import vrml.node.*;

public class TouchTranslator extends Script {
//CONSTANTS
  public final static boolean DEBUG = true;

//FIELDS
  SFString out;      //eventOut
  String outname;   //string sent to ObjMover; tells it that
               //this object was pushed

//PUBLIC METHODS
  public void initialize() {
    outname =
      ((SFString)getField ("outName")).getValue();
    dprint ("initializing Translator:"+outname);
    out = (SFString)getEventOut ("touched");
  }

  public void processEvent(Event evt) {
    dprint ("processing event in Translator:"+evt.getName());
    try {
      if (evt.getName().equals ("touchTime")) {
        dprint ("translating to "+outname);
        out.setValue(outname);
      }
    } catch (Exception e) {
      System.err.println
        ("Error in translator.processEvent():");
      e.printStackTrace();
    }
  }

  public void shutdown ()
  {dprint ("shutdown called in Translator");}

//PRIVATE METHODS
  private void dprint (String s)
  {if (DEBUG) System.out.println (s);}
}
```

---

```
/* WorldHandler.java
includes only WorldHandler class
deals with graphical interface and manipulating 3-D world
Here is the set-up of the 3-D world
  1) the back wall -- This is where the user starts, facing into the room.
    There is a control panel and a door leading to the superdiscussion (from
    which the user came) on the wall.
  2) the comment wall -- This will be to the left of the user when he enters
    It contains all the posted comments
  3) the subdiscussions -- This will be to the right of the user when he enters
    the room.  It will be a set of subdiscussion room prototypes, with a door
    leading to each subdiscussion room
  4) the objects -- The imported objects, which include images and 3-D objects,
    and in the future (hopefully), interfaces for playing audio clips and
    movie clips.  They are lined up in the middle of the room.
*/

import vrml.*;
import vrml.field.*;
import vrml.node.*;
import java.util.*;
import java.io.PrintStream;
import java.net.*;
import StringCode;
import ConfClient;
import StreamManager;
import Import;
import Subdiscussion;
import SuperDisc;
import Picture;
import Comment;
import ThreeDObj;
import ObjMover;

public class WorldHandler
{

// CONSTANTS
  protected static final boolean DEBUG = ConfClient.DEBUG;
  protected static final StringCode sc = ConfClient.sc;
  private static final float SPACEFORNEW = (float)5.5; //how far beyond the current
      //furthest import that the next import should be placed
  private static final float[] FIRSTNEWOBJPOS = {0,0,-3}; //position of
    //first object imported, if it isn't specified a position
// private static final float[] FIRSTNEWCOMMPOS = {0,0,0}; //position of
    //first comment posted
// private static final float[] INITDIMS = {0,0,0}; //default value
    //for dims
  private static final int X = 0;
  private static final int Y = 1;
  private static final int Z = 2;

// FIELDS
  //fields/events passed in from VRML world
  SFNode client;   //node for ConfClient
// SFNode world;   //VRML world to which objects can be added
// SFNode commentWall; //comment wall in VRML world
  MFNode addObj;   //eventOut used to add imported objects &
                   //subdiscussion prototypes to world
  MFNode removeObj; //eventOut used to remove them
  MFNode addDisc; //used to add subdiscussion prototypes
  MFNode removeDisc;
  SFBool setFrontView; //set to true to bind default ViewPoint
// MFNode addComm;   //used to add comments to comment wall
// MFNode removeComm; //used to remove them

  //fields used to keep track of state
  public String Disc;  //name of current discussion
  public String User;  //name of user for this world
  float[] nextObj;
    //coordinates ofwhere next object should be placed if no coordinates
    //are specified
// float[] nextComm=StreamManager copyDims(FIRSTNEWCOMMPOS);
```

```
    //coordinates of next comment
  float nextDisc = 0;  //where along wall with subdisc rooms
      //to put next subdisc.  Indicates position of right wall.
//  float dims[] = copyDims (INITDIMS);

  //fields for adding/removing objects
  Browser browser;  //browser in which vrml world is
//  Hashtable Imports = new Hashtable (2); //Imported images & 3-D objs
  Hashtable Images = new Hashtable (1); //imported images
  Hashtable vrmlObjs = new Hashtable (1);  //imported 3-D objects
  Hashtable Subdiscs = new Hashtable (1);  //subdisc prototypes
  Hashtable Comments = new Hashtable(3); //Comment windows
      //names of speakers hash to their commentwindows
  SuperDisc superDisc;

  //other fields
//  public ConfClient Client;
  public StreamManager output;
  ObjMover objMover;

// CONSTRUCTORS
//  public WorldHandler(ConfClient client, String disc, String user,
//    /*StreamManager bc_output,*/ StreamManager rq_output, Browser b,
//    MFNode ao, MFNode ro, MFNode ac, MFNode rc)
  public WorldHandler(String disc, String user,
    StreamManager rq_output, Browser b, SFNode cl,
    SFNode w, SFNode subdiscHolder, SFBool se, SFBool fv,
    SFColor chXCol, SFColor chYCol, SFColor chZCol,
    SFBool enXSen, SFBool enYSen, SFBool enZSen, SFNode a)
  {
    dprint("WorldHandler:init");

    //setting fields through variables passed in.
//    Client = client;
    User = user;
    output = rq_output;
//    broadcast_output = bc_output;
    browser = b;
    client = cl;
    setFrontView = fv;

    //setting fields to add & remove objects from world
    //world = w;
    addObj = (MFNode)((Node)w.getValue()).getEventIn ("addChildren");
    removeObj = (MFNode)((Node)w.getValue()).getEventIn ("removeChildren");

//    commentWall = cw;
//    addComm = (MFNode)((Node)commentWall.getValue()).getEventIn ("addChildren");
//    removeComm = (MFNode)((Node)commentWall.getValue()).getEventIn ("removeChildren");
```

```
  addDisc = (MFNode)((Node)subdiscHolder.getValue()).getEventIn ("addChildren");
  removeDisc = (MFNode)((Node)subdiscHolder.getValue()).getEventIn ("removeChildren");

  //create ObjMover
  objMover = new ObjMover (se, chXCol, chYCol, chZCol, a,
                   enXSen, enYSen, enZSen, cl, b, output);

  reset (disc);
  dprint("WorldHandler:done with init");
} // end WorldHandler constructor

public void reset (String disc)
// sets disc name and removes objects from world
// used upon construction, and when user is switching discussions
// used when CLEARDESKTOP is sent from server
{
  dprint ("resetting world");
  Disc = disc;

  dprint ("removing images from world");
  for (Enumeration enum = Images.elements();
     enum.hasMoreElements();)
   ((Import)enum.nextElement()).remove(removeObj,browser);
  Images = new Hashtable(2);

  dprint ("removing 3-D objects from world");
  for (Enumeration enum = vrmlObjs.elements();
     enum.hasMoreElements();)
   ((Import)enum.nextElement()).remove(removeObj,browser);
  vrmlObjs = new Hashtable(2);

  dprint ("removing subdiscussions from world");
  for (Enumeration enum = Subdiscs.elements();
     enum.hasMoreElements();
     ((Subdiscussion)enum.nextElement()).remove(removeDisc));
  Subdiscs = new Hashtable(1);

  if (superDisc != null) {
   dprint ("removing superdiscussion from world");
   superDisc.remove(removeObj,browser);
   superDisc = null;
  }

  dprint ("removing comments from world");
  for (Enumeration enum = Comments.elements();
     enum.hasMoreElements();
     ((Comment)enum.nextElement()).remove(removeObj,
        browser));
  Comments = new Hashtable(3);
```

```
    nextObj = output.copyDims(FIRSTNEWOBJPOS);
    nextDisc = 0;
//   nextComm = copyDims(FIRSTNEWCOMMPOS);
//   dims = copyDims (INITDIMS);
    objMover.reset();
    setFrontView.setValue(true);
  } //reset

  public void processEvent (Event evt) {
  //an event occurred in the VRML world
    dprint ("processing event: "+evt.getName());
    if (evt.getName().equals("doorClicked"))
      //door for superdiscussion was clicked
      //in the future, code below will be used
      output.send (StringCode.CHANGEDISC,superDisc.title());
 /* because Scripts can't be added dynamically (so it seems)
  TouchTranslator scripts can't be added to distinguish between
  clicks of different doors.  Therefore, for now, doorClicked
  always corresponds to the superdiscussion door.  In the future,
  doorClicked should be an SFString which will contain the title
  of the discussion whose door was clicked.
      {
       String title = ((ConstSFString)evt.getValue()).getValue();
       output.send (StringCode.CHANGEDISC,title);
      }
  */

    else {
      dprint ("event not processed; passing on to ObjMover");
      objMover.processEvent(evt);
    }
  }

  public void setDisc (String d)
  {
    dprint ("setting discussion to "+d);
    Disc = d;
  }

  public void sendSize()
  //sends size of VRML world to user
  {objMover.sendSize();}

  public void addComment (String id, String text, String coordstr)
    throws InvalidVRMLSyntaxException
  //place Comment at given coordinates
  {
    float[] coords;
    if (coordstr.equals(sc.EMPTYFIELD)||coordstr.equals(sc.REPLY))
       //set to default position
      coords = StreamManager.copyDims(nextObj);
    else coords = output.parseTriple(coordstr);
    addComment (id, text, coords, coordstr.equals(sc.REPLY));
  }  //addComment

  public void addComment (String id, String comment, float[] pos,
                   boolean reply)
    throws InvalidVRMLSyntaxException
  //adds comment to world
  //id contains speaker & an id number to distinguish it from
  //    other comments he's made
  //if reply is true, sends message to server telling it where
  //    object is placed
  {
    dprint ("adding comment to world: "+id+','+comment);

    //extract speaker from id
    String speaker = output.parseCommIdGetSpkr (id);
    //create comment & add it to VRML world
//    Comment newComm = new Comment (speaker, comment, pos[X],
//changed so that user can distinguish comments if he wants to
//  remove them
      Comment newComm = new Comment (id, comment, pos[X],
            pos[Y], pos[Z], browser, addObj, client);
//    if (nextComm[0]<x+SPACEFORNEW) nextComm[0]=x+SPACEFORNEW;

    updateNextPos(pos);

    Comments.put (id, newComm);
    objMover.addComment(id, newComm.getInfo());
    if (reply) output.send
          (sc.COMMENTSTRING,id,output.unparseTriple(pos));
//    addComm.setValue(newComm.Node());
  }

  public void addImage (String title, String url, String coordstr)
    throws InvalidVRMLSyntaxException
  //place Image at given coordinates
  {
    float[] coords;
    if (coordstr.equals(sc.EMPTYFIELD) ||
       coordstr.equals(sc.REPLY))
       //set to default position
       coords = StreamManager.copyDims(nextObj);
    else coords = output.parseTriple(coordstr);
    addImage (title, url, coords, coordstr.equals(sc.REPLY));
  }  //addImage

  public void addImage (String title, String url, float[] pos,
                   boolean reply)
```

```
 throws InvalidVRMLSyntaxException
//adds 2-D Image to world
//if reply is true, tells server of pos at which image
//  is placed
{
 dprint ("adding image to world:"+title+','
       +output.unparseTriple(pos));
 try {
  Picture newImage = new Picture (title, url, pos[X],
         pos[Y], pos[Z], browser, addObj, client);
  updateNextPos(pos);

  //update imports list and ObjMover
  Images.put (title,newImage);
  objMover.addImage(title,newImage.getInfo());
  if (reply) output.send
      (sc.PICWINSTRING,title,output.unparseTriple(pos));
 } catch (InterruptedException e)
 {StreamManager.showError(e);}
}    //addImage (String,String,int,int,int)

public void add3DObj (String title, String url, String coordstr)
 throws InvalidVRMLSyntaxException
//place 3-d object at given coordinates
{
 float[] coords;
 if (coordstr.equals(sc.EMPTYFIELD) ||
    coordstr.equals(sc.REPLY))
    //set to default position
    coords = StreamManager.copyDims(nextObj);
 else coords = output.parseTriple(coordstr);
 add3DObj (title, url, coords, coordstr.equals(sc.REPLY));
}  //add3DObj

public void add3DObj (String title, String url, float[] pos,
               boolean reply)
 throws InvalidVRMLSyntaxException
//adds 3-D object to world
//if reply is true, sends message to server telling it where
//   object is placed
{
 dprint ("adding 3-D object to world:"+title+','
       +output.unparseTriple(pos));
 ThreeDObj newObj = new ThreeDObj (title, url, pos[X], pos[Y],
                   pos[Z], browser, addObj, client);
 updateNextPos(pos);
 vrmlObjs.put (title,newObj);
 objMover.add3DObj(title,newObj.getInfo());
 if (reply) output.send
       (sc.OBJSTRING,title,output.unparseTriple(pos));
 }    //add3DObj (String,String,int,int,int, boolean)

 public void moveImport (String type, String id, String newposstr)
 //moves import (called when another user moves the object)
 // type is whether it's a picture, 3-D object, etc.  This field is not
 // currently used
 //id identifies which object was moved.
 {
   dprint ("moving import in world: "+id+','+newposstr);
   Node[] moved = ((Import)findImportList(type).get(id)).
                                        getNode();

   //parsing newpos
   float[] newpos = output.parseTriple(newposstr);

/*  NOW TAKEN CARE OF BY ObjMover
   //manipulating VRML world
   SFVec3f pos = (SFVec3f)(moved[0]).getExposedField("position");
   pos.setValue(newpos[X],newpos[Y], newpos[Z]);
*/
   objMover.moveObj (type,id,newpos);

   //updating nextObj -- need to change default pos so next
   //import isn't placed on top of this one
   updateNextPos(newpos);
 }  //moveObj

 public void removeImport (String type, String id)
 //removes Import from world
 //type = comment, picture, etc.
 //id = identifier (title or speaker&id
 {
   dprint ("removing import in world:"+id);

   //simultaneously remove from Hashtable and world
   ((Import)findImportList(type).remove(id)).
      remove(removeObj,browser);

   //remove from list in objMover
   objMover.removeImport (type,id);
 }

 public void addSubDisc (String discname, String negdimstr,
              String posdimstr)
 //adds subdiscussion prototype to world
 //negdimstr = parsable ordered triple that contains the
 //    smallest/most negative coordinates occupied by any
 //    object in the discussion
 //posdimstr = same, but largest coordinates
 {
  dprint ("adding subdisc in world: "+discname);

  //parsing dims to get dimensions of room
  float[] negdims = output.parseTriple(negdimstr);
  float[] posdims = output.parseTriple(posdimstr);

  try {
    Subdiscussion newdisc = new Subdiscussion
        (discname,addDisc,client, browser, nextDisc,
        negdims,posdims);
    Subdiscs.put (discname, newdisc);
    nextDisc -= newdisc.getWidth();
  } catch (Exception e) {StreamManager.showError(e);}
} //addSubDisc

public void addSuperDisc (String discname)
//notifies world of where superdiscussion door should lead
{
  dprint ("adding superdisc in world: "+discname);

  //check to make sure there's not already a superdisc
  if (superDisc != null) {
    System.err.println ("Error: trying to add"
      +"superdiscussion when one already exists.");
    return;
  }      //if SuperdiscTitle

  //check was ok; proceed
  try {
    superDisc = new SuperDisc (discname,addObj,client,
                          browser);
  } catch (Exception e) {StreamManager.showError(e);}
} //addSuperDisc

public boolean addProto (String importtype, String discname,
        String url, String coords)
//adds ImportProto to given subdisc
//discname = name of subdiscussion to which to add prototype
//importtype = comment, image, etc.
//coords = position, relative to subdiscs's origin, at which
//    to place prototype
//returns true if addition is successful; false otherwise
{
  dprint ("adding prototype to subdiscussion:"+discname);
  try {
    ((Subdiscussion)Subdiscs.get(discname)).
      addProto(importtype,url,output.parseTriple(coords));
  } catch (NullPointerException e) {
    System.err.println ("Error:Subdiscussion does not exist");
    return false;
    }
    return true;
  }     //addProto

//PRIVATE FUNCTIONS
private Hashtable findImportList (String list)
//returns Comments, Images, or vrmlObjs
{
  if (list.equals(sc.COMMENTSTRING)) return Comments;
  if (list.equals(sc.PICWINSTRING)) return Images;
  if (list.equals(sc.OBJSTRING)) return vrmlObjs;
  else System.err.println ("Error: Import list type not found:"
       +list);
  return null;
}

private void updateNextPos (float[] newPos)
//requires: newPos has 3 elements
//an object has been added or moved to newPos
//change nextObj
//nextObj is recorded so that the next import will not be
//  placed in the same position as an already existent import
{
  if (nextObj[X]>newPos[X]-SPACEFORNEW)
    nextObj[X]=newPos[X]-SPACEFORNEW;
}       //updateNextObj

private void dprint (String str)
{if (DEBUG) System.out.println (str);}
} // end WorldHandler
```

## IV. Shared Classes

```
/* InputLoop.java
 includes InputLoop class

 *         This class is a loop that continually checks an input stream to see if
 *         any bytes have come in.  It parses incoming messages by looking for a
 *     "start" byte and a "stop" byte.  It then processes these messages as
 *         specified in the process_message() method.  Note that in order to be
 *         useful, InputLoop should be subclassed and process_message() redefined
           This class is used both on the client side and server side.
 */


import java.lang.StringBuffer;
import java.io.DataInputStream;
import java.net.SocketException;
```

```
public abstract class InputLoop extends Thread {
// public abstract class InputLoop {

 // CONSTANTS
 public static final byte START = 1;
 public static final byte STOP = 2;
 public static final int MAXBYTES = 100;
 protected static final StringCode sc = new StringCode();
 protected static final boolean DEBUG = true;

 // FIELDS
 public boolean go=true; // in case loop needs to be stopped without
                          //terminating the Thread
 protected DataInputStream input;
// int byte_count=0; // to make sure that InputLoop doesn't hog processor by limiting the number of bytes
(characters) that are read in one "cycle"
// protected ServerConnection conn;

 // METHODS

 public InputLoop(DataInputStream in) {
   input = in;
//   conn = c;
   this.setDaemon(true); // may not be necessary
   this.setPriority(Thread.MIN_PRIORITY); // this should be fine, since input can accumulate in the
buffer without any problems
 } // end InputLoop constructor

 public void run()
// public void run(ServerConnection conn)
 {
   int available = 0; // the number of bytes in the buffer
   boolean reading = false; // to make sure that if for some reason a start or stop byte is missed only one
"message" is messed up
   byte c;
   StringBuffer message = new StringBuffer();
   try {
     int byte_count=0; // to make sure that InputLoop doesn't hog processor
                        // by limiting the number of bytes (characters) that are read
                        // in one "cycle"
   while (go) {
//     displayRunning();
     byte_count = 0;
         available = input.available();
         if (available > 0) {
           for (int i = 0; i < available; i++) {
       if (byte_count >= MAXBYTES) yield();
//         if (byte_count >= MAXBYTES) conn.yield();
             c = input.readByte();
```

```
          byte_count++;
              if (c == START) {
                message = new StringBuffer();
                reading = true;
              }
              else {
                if (c == STOP) {
                  if (reading) {
                    reading = false;
                    process_message(message.toString());
             }
                }
                else {
                  if (reading) message.append((char) c);
                }
              }                    //if c else
            }                      //for int
          }                        //if available >0
          else yield();          //let other threads run
//        else conn.yield();
      }                  //while go
      sleep(1000);
    }  catch(Exception e) {
          go = false;
          if (e instanceof SocketException)
            handleSocketException ();
          else StreamManager.showError(e);
    }                  //try/catch
  } // end run

  protected abstract boolean process_message(String message);

  protected abstract void handleSocketException();
/*
  //should be overridden by subclasses
  {this.stop();}
*/

//  protected abstract void displayRunning(); //temporary
} // end InputLoop
```

```
/* StreamManager.java
Contains StreamManager class only.

Used to make sure that multiple messages are not being sent through same
stream over the Internet at same time.
*/

import java.io.PrintStream;
import java.util.*;
import StringCode;

public class StreamManager {
//CONSTANTS
  private final static boolean DEBUG = true;
  private final static StringCode sc = new StringCode();
//FIELDS
  PrintStream output;

//CONSTRUCTORS
  public StreamManager (PrintStream o) {output=o;}

//NONSTATIC METHODS
  public synchronized void send (String message)
  {
    System.out.println("stream manager sending:" + message);
    output.write(InputLoop.START);
    output.print(message);
    output.write(InputLoop.STOP);
  }  //send

  public void send (String m1, String m2)
  {send (unparseMessage (m1,m2));}

  public void send (String m1, String m2, String m3)
  {send (unparseMessage (m1,m2,m3));}

  public void send (String m1, String m2, String m3, String m4)
  {send (unparseMessage (m1,m2,m3,m4));}

  public void send (String m1, String m2, String m3, String m4, String m5)
  {send (unparseMessage (m1,m2,m3,m4,m5));}

  public void send (String m1, String m2, String m3, String m4, String m5,
                          String m6)
  {send (unparseMessage (m1,m2,m3,m4,m5,m6));}

//STATIC METHODS
//parse and unparse functions
  public static String unparseMessage (String s1, String s2)
  {return s1+sc.MESSAGEDIVIDER+s2;}

  public static String unparseMessage (String s1, String s2, String s3)
  {return s1+sc.MESSAGEDIVIDER+s2+sc.MESSAGEDIVIDER+s3;}

  public static String unparseMessage (String s1, String s2, String s3,
                                  String s4)
  {return s1+sc.MESSAGEDIVIDER+s2+sc.MESSAGEDIVIDER+s3+sc.MESSAGEDIVIDER+s4;}

  public static String unparseMessage (String s1, String s2, String s3,
                                          String s4, String s5)
  {return s1+sc.MESSAGEDIVIDER+s2+sc.MESSAGEDIVIDER+s3+sc.MESSAGEDIVIDER+s4
                   +sc.MESSAGEDIVIDER+s5;}

  public static String unparseMessage (String s1, String s2, String s3,
                                          String s4, String s5, String s6)
  {return s1+sc.MESSAGEDIVIDER+s2+sc.MESSAGEDIVIDER+s3+sc.MESSAGEDIVIDER+s4
                   +sc.MESSAGEDIVIDER+s5+sc.MESSAGEDIVIDER+s6;}

  public static StringTokenizer parseMessage (String s)
  {return new StringTokenizer (s,sc.MESSAGEDIVIDER,false);}

  public static float parseFloat (String s)
    throws NumberFormatException
  {return (new Float(s)).floatValue();}

public static String unparseTriple  (float[] triple)
//triple represents an ordered triple
//simply places IDDIVIDERS between each element of the triple
{return unparseTriple (triple[0],triple[1],triple[2]);}

public static String unparseTriple (float x, float y, float z)
{
  return (Float.toString(x)+sc.IDDIVIDER+y+sc.IDDIVIDER+z);
}      //unparseTriple

public static float[] parseTriple (String triplestr)
//returns ordered triple, assuming triplestr uses IDDIVIDER to
//  separate elements of triple
{
  StringTokenizer tok =
    new StringTokenizer (triplestr,sc.IDDIVIDER,false);
  float[] returnval = {parseFloat(tok.nextToken()),
                  parseFloat(tok.nextToken()),
                  parseFloat(tok.nextToken())};
  return returnval;
}  //parseTriple

public static String unparseList (Enumeration enum)
//returns elements of enum, an enumeration of strings
{
  if (!enum.hasMoreElements()) return sc.EMPTYFIELD;
  StringBuffer returnstr =
         new StringBuffer(enum.nextElement().toString());

  while (enum.hasMoreElements())
    returnstr.append(sc.PARAMDIVIDER+enum.nextElement().toString());

  return returnstr.toString();
}                    //unparseList
```

```
public static Vector parseList (String list)
//parse into Vector of Strings
{
 Vector returnval = new Vector (2);
 if (list.equals(sc.EMPTYFIELD))
   //then return an empty vector
   return returnval;

 StringTokenizer tok = new StringTokenizer (list,sc.PARAMDIVIDER,false);

 while (tok.hasMoreElements())
   returnval.addElement (tok.nextToken());

 return returnval;
}                         //parseList

public static String unparseCommId (String spkr, int idnum)
{return spkr+sc.IDDIVIDER+idnum;}

public static String parseCommIdGetSpkr (String id)
{
 StringTokenizer tok =
   new StringTokenizer (id,sc.IDDIVIDER,false);
 return tok.nextToken();
}

//other static functions
 public static String restOf (StringTokenizer tok)
 //returns concatenation of rest of tokens in tok, with MESSAGEDIVIDERs
 //        between tokens
 {
   //initialize it, just in case tok has no tokens
   StringBuffer str = new StringBuffer();
   while (tok.hasMoreTokens())
     str.append(sc.MESSAGEDIVIDER+tok.nextToken());
   if (DEBUG) System.out.println ("Result from restOf: "+str);
   return str.toString();
 }

 public static String findShortName (String name)
 //returns name without spaces
 {
/* using this procedure puts unnecessary restrictions on what
the user can name objects, discussions, and themselves.  Before,
I used this procedure because I thought having spaces in file
names was a problem.  Apparently, it's not.

   int namelen = name.length();
   StringBuffer namebuff = new StringBuffer();
```

```
    char nextchar;

    for (int charnum=0; charnum<namelen; charnum++)
      if ((nextchar=name.charAt(charnum))!=' ')
        namebuff.append(nextchar);
//   System.out.println ("shortname of "+name+':'+namebuff);
    return namebuff.toString();
*/
    return name;
  }  //findShortName

  public static boolean similarName(String name, Vector list,
        boolean identicalsOk)
  //returns true if name has same shortName as a name in the list
  //exception: if name is identical to name of previous user, returns
  //identicalsOk
  {
    String shortname = findShortName(name);
    String nextname;
    for (Enumeration enum = list.elements();
       enum.hasMoreElements();) {
      nextname = (String) enum.nextElement();
      if (nextname.equals(name)) return identicalsOk;
      if (findShortName(nextname).equals (shortname))
        return true;
    }
    return false;  //no similar names found
  }  //similarName

  public static float[] copyDims (float[] original)
//original must have exactly 3 elements
//makes copy of original and returns it.
//original is an array of three floats
//with just normal assignment (with =), original changes when copy changes,
// but with this procedure, that doesn't happen.
{
  float[] returnArray = {original[0],original[1],original[2]};
  return returnArray;
}

public static void showError (Exception e)
{
  System.err.println ("Error:");
  e.printStackTrace();
}
} //Streammanager
```

---

```
/* StringCode.java
Contains StringCode class.
This class contains a number of constants that are shared by both
```

```
the server & client.  To see how most of these are handled, see
WSInLoop in ConferenceServer.java, PSInLoop in PromptServer.java,
WorldInLoop in ConfClient.java, and PromptInLoop in PromptFrame.java.
*/
import StreamManager;

public class StringCode {
  public StringCode() { }

/***********************SERVER INFO***********************************/
  public static final String HOSTNAME = "heinlein.media.mit.edu";
  public static final int PROMPT_PORT = 7000;
  public static final int WORLD_PORT = 7001;
  public static final String DEFAULTDISC = "Main"; //default discussion
      //into which a user gets put

/************STRINGS USED TO DIVIDE COMPONENTS OF VARIOUS THINGS********/
/*Most of these dividers are typable characters.  This convention was used
so that debugging would be easier.  In the final version of the program,
though, all of the dividers should be untypable (e.g., "\1","\2")
*/
  //USED IN MESSAGES BETWEEN CLIENT AND SERVER
  //to divide main components of message between server & client
  public static final String MESSAGEDIVIDER = "|";

  //to separate window identifiers
  public static final String IDDIVIDER = "~";

  //to separate parameters of layouts for a particular window
  public static final String PARAMDIVIDER = "`"; //used to separate
                        //window parameters in window layout

  //FOR FILES
  //to separate records (see Record.java) in history files
  public static final String HISTDIVIDER = "-\0------------";
  public static final String DONESTRING = "DONE";

  //**************CODES SENT FROM SERVER****************
/*These codes always begin messages sent from the server to the client.*/
  //FOR HANDLING NEW USERS
  public static final String NEWUSERSTRING = "NEWU";
  public static final String GETNAMESTRING = "GNME";
  public static final String NEWNAMESTRING = "NNME";
  public static final String GETPWDSTRING = "GPWD";

  //WINDOWS
  public static final String NEWWINSTRING = "NEWW";
  public static final String CHATSTRING = "Chat";
  public static final String HISTWINSTRING = "Hwin";
  public static final String ERRWINSTRING = "Ewin";
public static final String PROMPTWINSTRING = "Prwn";
public static final String SETTEXT = "STXT";
public static final String CLOSECHAT = "CCHT";
public static final String REMOVECOMM = "RCOM";

//IMPORTS
public static final String LAYOUTSTRING = "LYOT";
public static final String COMMENTSTRING = "Comm";
public static final String PICWINSTRING = "Pwin";
public static final String OBJSTRING = "3Obj";
public static final String OBJMOVEDSTRING = "OMVD";
public static final String REMOVEIMPORT = "RIMP";
public static final String REPLY = "reply!";
    //used when object is added to world, and server wants
    //a reply as to where the object is placed

//EXCEPTIONS
public static final String PERSONCAPSTRING = "PCAP";
public static final String DISCFULLSTRING = "DFUL";
public static final String ACTIVENAMESTRING = "ANME";
public static final String MAXDISCSTRING = "MDSC";

//CONCERNING DISCUSSIONS
public static final String ADDSUBDISC = "ASUB";
public static final String ADDSUPERDISC = "ASUP";
public static final String ADDPROTO = "APTO";
public static final String ASKFORSIZE = "AFSZ";
  //used to ask for current discussion size
  //called after something is just imported.  Client doesn't
  //automatically return new size because if it did, then
  //every client would have to.  So ASKFORSIZE causes only
  //one client to return the new size.
public static final String CLEARDESKTOP = "CDTP";
public static final String COMMENTLIST = "CLST";
  //send list of title of comments for current discussion
  //so user can choose when he wants to remove something

//*********STRINGS SENT FROM CLIENT************
//FOR NEW USERS
  //NEWUSERSTRING (see above)
public static final String USERNAMESTRING = "NAME";
public static final String CREATEPERSTRING = "CPER";

//WINDOW REQUESTS
//public static final String NEWCHATSTRING = "NCHT";
        //HISTWINSTRING (see above)
public static final String SENDCHATSTRING = "SCHT";
        //CLOSECHAT (see above) (not used by DDS yet)
  //SETTEXT (see above)
  //CONCERNING IMPORTS
  public static final String SENDCOMMSTRING = "COMM";
          //PICWINSTRING (see above)
          //OBJSTRING (see above)
          //OBJMOVEDSTRING (see above)
          //REMOVEIMPORT (see above)

  //CONCERNING DISCUSSIONS
  public static final String NEWDISCSTRING = "NDSC";
  public static final String CHANGEDISC = "CDSC";
  public static final String DISCNAMESTRING = "DSNM";
  public static final String WORLDSIZE = "WSZE";
      //size of world corresponding to current discussion

  /*********************OTHER**********************/
  public static final String EMPTYFIELD = " "; //used instead of "" because
              //StringTokenizer does not handle "" well
  public static final int MAXDISCSIZE = 20;  //maximum valid value for a
                                              //discussion capacitY
  public static final String DEFAULTDISCDIMS =
   StreamManager.unparseTriple(0,0,0);
   //default dimensions of discussions
}
```

## V. Other Code

```
<!-- DDS.html -->
<!-- Main HTML file for DDS -->
<!-- includes 3 frames: one for queries (interface), one for -->
<!--      histories (history), and one for the VRML world (world) -->
<header>
<title> Design Discussion Space </title>
</header>

<frameset rows=0,* border=0>
<noframes>
<center>
Your browser does not support frames.

<applet>
It is also possible that your browser does not support Java code.  Therefore,
this application will not function properly.<br><br>
Try downloading a frame-capable browser such as <a href="http://www.netscape.com/">
Netscape</a>.<br><br>
</noframes>

<frame name=prompt src=PromptFrame.html marginwidth=0 marginheight=0
     scrolling=auto>
```

```
<frame name=world src=client/DDS.wrl marginwidth=0 marginheight=0
      scrolling=no>
</frameset>
```

---

```
#VRML V2.0 utf8
#main wrl file for DDS
#contains base form of DDS world and ConfClient script
#ConfFrame class adds and removes objects from world
#         through appropriate routes

##################external prototypes############
EXTERNPROTO DiscWall [
  field SFVec3f size
  field SFFloat transparency
]
"prototypes/DiscWall.wrl"

# the following are defined so that doors, images,etc.
#         can be added/removed in the world
# they have been commented out because createVrmlFromString() doesn't
# recognize prototypes
#EXTERNPROTO CommentWin [
#  field MFString speaker
#  field MFString quote
#  field SFVec3f position
#]
#"prototypes/CommentWin.wrl"
#
#EXTERNPROTO Image [
#  field MFString title  #label for this image
#  field SFNode image  #ImageTexture node
#  field SFVec3f position  #relative to world
#  field SFVec3f imageSize  #size of canvas for image
#  field SFVec3f titlePos #center of title bar, relative to image
#  field SFVec3f titleSize #size of title bar
#]
#"prototypes/Image.wrl"
#
#EXTERNPROTO 3DObj [
#  field MFString title  #label for this image
#  field MFString url  #ImageTexture node
#  field SFVec3f position  #relative to world
#  field SFVec3f titlePos #center of title bar, relative to object
#  field SFVec3f titleSize #size of title bar
#]
#"prototypes/3DObj.wrl"
#
#EXTERNPROTO Door [
#  field SFString description
#  field SFVec3f translation
#  field SFRotation rotation
#  field SFFloat transparency
#  field MFString destination
#]
#"prototypes/Door.wrl"

######################navigation infos#################
NavigationInfo { type "FLY" }
DEF EXAMNAV NavigationInfo { type "EXAMINE" }

########################view points####################
DEF FRONTVIEW Viewpoint {
 position 0 0 4
 description "Starting viewpoint"
}

Viewpoint {
 position 0 0 0
 orientation 0 1 0 3.14
 description "To superdiscussion"
}

# comment wall doesn't exist any more!!
#Viewpoint {
#  position 0 0 -5
#  orientation 0 1 0 1.571
#  description "Comment Wall"
#}
########################base form of world###########
DEF WORLD Group { } # used to add children to world

DEF DISCHOLDER Transform {
 translation 10 -5 4.75
 rotation 0 1 0 -1.57
}

#DEF COMMENTWALL Transform {
#  bboxCenter 5 0 0
#  bboxSize 20 10 .5
#  translation -9.7 0 0
#  rotation 0 1 0 1.571  #perpendicular rotation
#  children [
#    Transform {
#      translation 0 0 -.3   # set back so that comments added will be in
#                            # front of wall
#      children DEF OPAQWALL DiscWall { } # opaque wall
#    }  # Transform
#    Transform {
#      translation 10 0 -.3
#      children USE OPAQWALL
#    } #Transform
#  ] #children
#} #Transform (COMMENTWALL)

Transform {                    #back wall
  bboxCenter -5 0 0
  bboxSize 20 10 .5
  translation -5 0 4.75
  children [
    DEF OPAQWALL DiscWall{ }     # opaque wall
    Transform {
      translation 10 0 0
      children USE OPAQWALL
    } #Transform
  ] #children
} #Transform (back wall)

#AXES FOR MOVING OBJECTS (DOESN'T SHOW UP IN BASE FORM)
Switch {        #so it won't show up
  choice
    DEF AXES Group {
      children [
        Group {
          children [
            DEF ZTOUCH TouchSensor { }
            Shape {
              appearance Appearance {
                material
                  DEF ZAXIS_MAT Material {
                    diffuseColor .1 .1 .9
                  }
              }    #appearance
              geometry DEF AXIS Box {
                size .1 .1 1.5
              }
            }       #Z_AXIS
          ]    #children = ZTOUCH & ZAXIS
        }     #Group with z axis stuff
        Transform {     # y axis
          rotation 1 0 0 -1.57
          children [
            DEF YTOUCH TouchSensor{ }
            Shape {
              appearance Appearance {
                material
                  DEF YAXIS_MAT Material {
                    diffuseColor .1 .1 .9
                  }
              }    #appearance of Y-AXIS
              geometry USE AXIS
          }       #Y_AXIS
        ]    #children = YTOUCH and Y_AXIS
      }     #Transform with y axis stuff
      Transform {
        rotation 0 1 0 1.57
        children [
          DEF XTOUCH TouchSensor{ }
          Shape {
            appearance Appearance {
              material
                DEF XAXIS_MAT Material {
                  diffuseColor .1 .1 .9
                  }
            }    #appearance of X_AXIS
            geometry USE AXIS
          }       #X_AXIS
        ]    #children = XTOUCH and X_AXIS
      }     #Transform with x axis stuff
    ]    #children of Group in choice
  }      #Group
}      #Switch

##################scripts#######################
DEF CONFCLIENT Script {
  url "ConfClient.class"
  mustEvaluate TRUE
  directOutput TRUE

#  eventOut MFNode addObj          # used to add & remove imported objects,
#  eventOut MFNode removeObj    #   subdisc prototypes, & doors
#  eventOut MFNode addComm      # used to add & remove comment windows
#  eventOut MFNode removeComm

  field SFNode thisNode USE CONFCLIENT
  field SFNode world USE WORLD
#  field SFNode commentWall USE COMMENTWALL
  field SFNode subDiscHolder USE DISCHOLDER
  field SFNode axes USE AXES

  eventIn SFString titleClicked  #which title bar was clicked
  eventIn SFTime xClicked        # when x-axis is clicked
  eventIn SFTime yClicked        # y axis
  eventIn SFTime zClicked        #z
  eventIn SFVec3f translation_changed   #when object is moved
#  eventIn SFString doorCliecked  #which door was clicked
  eventIn SFTime doorClicked

  eventOut SFBool setExaminer  # changes NavigationInfo binding
  eventOut SFBool frontView  # changes ViewPoint to default
  eventOut SFColor changeXColor #changes emissiveColors of axes
  eventOut SFColor changeYColor
  eventOut SFColor changeZColor
  eventOut SFBool enableXPSensor  # enables PlaneSensors
  eventOut SFBool enableYPSensor
  eventOut SFBool enableZPSensor
}

########################routes###################
# ROUTE CONFCLIENT.addObj TO WORLD.addChildren
# ROUTE CONFCLIENT.removeObj TO WORLD.removeChildren
# ROUTE CONFCLIENT.addComm TO COMMENTWALL.addChildren
# ROUTE CONFCLIENT.removeComm TO COMMENTWALL.removeChildren

# let CONFCLIENT change browsing configurations
ROUTE CONFCLIENT.setExaminer TO EXAMNAV.set_bind
ROUTE CONFCLIENT.frontView TO FRONTVIEW.set_bind

# routes to translator scripts
ROUTE XTOUCH.touchTime TO CONFCLIENT.xClicked
ROUTE YTOUCH.touchTime TO CONFCLIENT.yClicked
ROUTE ZTOUCH.touchTime TO CONFCLIENT.zClicked
```

---

```
<!-- PromptFrame.html -->
<!-- Used to run PromptFrame applet, which lets the client communicate -->
<!-- with the server -->

<html>
<head>
<title>Promptframe of DDS</title>
</head>
<body>
<applet code="PromptFrame.class" codebase=promptframe
  vspace=0 hspace=0>
<param name=WIDTH value=400>
<param name=HEIGHT value=250>
</applet>
</body>
</html>
```