

CS 350 – Computer/Human Interaction

Fall 2006 – Visual Basic Prototype: A Chat Program

Out: September 15

Due: October 2

While a CGI web-based message board is interesting, your software company has decided that they would like to see a prototype chat program modeled after ICQ or AOL Instant Messenger. These chat programs also are similar to the Unix **talk** command, and allow real-time one-to-one, one-to-many, or many-to-many chatting via the Internet.

The Task

- Create a prototype interface for a chat program using Visual Basic 2005. The instructor will be happy to assist in discovering the capabilities of the VB environment.
- The prototype is to demonstrate the feasibility of such a product, and is mainly to demonstrate the chat interface. As such, the prototype does not need to actually communicate via the Internet. A simple way to allow communication between two running prototypes is to write files into the same directory (e.g., login into your account on two different machines and write to a network drive - I: on ACENET, H: in the CS Lab), and look into that directory often to see if any new messages have appeared. (Although the VB executable must be located on the local machine to be trusted to write files, the data files that it writes may be on a network drive.)
- Furthermore, the prototype need only allow one-to-one communication, although you can allow one-to-many or many-to-many communication if you so desire. There is no need to implement a login mechanism; it is the chatting itself which is the focus of the design.
- You should provide some feedback when sending messages, so the user knows the message has been sent, and appropriate error messages if something goes wrong, or some vital information is omitted. You will have to decide on how messages are displayed. You may wish to have the program automatically check for new messages every few seconds, or you might want the user to press a button to load all new messages.
- The information on a particular message could be stored into a file, one message per file, all messages in a single file, or not at all, but how the messages are transmitted is entirely up to you. In any case, store these messages separate from your previous prototype's information. If you do use files, be aware that file names will have to be generated in a similar fashion to the previous prototype. Also, you may wish to delete old messages, but this is up to you.

What to submit

All Visual Basic source code files of your project (that is, the files with the **.vb** extension, both the code files you write **and** the designer-generated files) are to be zipped (Windows) or tarred (Unix) and submitted as an attachment via email to the instructor.

Hand in the following items:

- A signed statement that the work you are submitting is your own (see [Project Overview](#))
- A description of the program structure and instructions on how to install your program, including any machine specific code that may need to be changed by the instructor to run your program (e.g., directory names).
- Hardcopy printouts of your *well-commented* Visual Basic code files. **Do not print out the designer-generated files.**
- A discussion about why you chose to organize the interfaces and storage the way you did, focusing on task analysis and activity design.

Grading

Prototypes will be graded in the following manner:

- 25 points - prototype demonstrates an interface that meets the above specification
- 5 points - aesthetics and usability
- 10 points -discussion

More Visual Basic notes

Classes that may be helpful include:

- **System.IO.Directory** - a class of shared methods for manipulating directories. Of particular interest is the **GetFiles** method, which can be called with a directory name as a string and returns an array of strings that are the file names.
- **System.DateTime** - a class defining DateTime objects. Properties of interest might include **Now** (a shared method that returns a DateTime object with the current date and time) , **Year**, **Month**, **Day**, **Hour**, **Minute**, **Second**.
- **System.Timers.Timer** - a class defining Timer objects that will raise an ElapsedTime event after a set interval.

Web links to the MSDN pages that define these classes as well as a link to the MSDN pages for the entire .NET library are given in the on-line [Supplemental Reading References](#). Several of the member function entries have code examples on how to use them.