(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
20 November 2003 (20.11.2003)

PCT

(10) International Publication Number
WO 03/096141 A2

(51) International Patent Classification⁷: G06F

(21) International Application Number: PCT/IB03/02689

(22) International Filing Date: 14 May 2003 (14.05.2003)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/381,060 14 May 2002 (14.05.2002) US

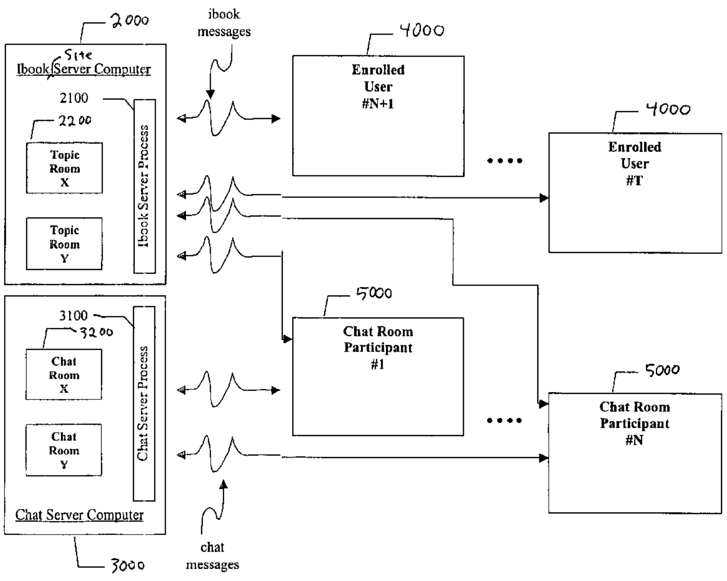
(71) Applicants and
(72) Inventors: **REYNOLDS, Brian** [GB/JM]; Kimmage, Main Street, Hopewell, Hanover (JM). **CONWAY, Mark, J.** [GB/GB]; #8 The Manlowes, Newbury RG14 7AY (GB).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NI, NO, NZ, OM, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

(54) Title: INTERACTIVE WEB COLLABORATION SYSTEMS AND METHODS



(57) **Abstract:** A topic room is provided in which one or more individuals or other entities may collaborate on topics of mutual interest. Multiple individuals or participants may use the topic room to communicate in real or non-real time and may work together to create, browse, modify, comment on, and perform any other suitable action on content. A chat room within the topic room receives, records, and transmits the communications and all activity in the topic room to all participants as messages. Client processes at a participant's user equipment may listen to the messages and take particular actions. For example, one participant can follow another participant as that participant browses through material by using the messages received from that participant.

INTERACTIVE WEB COLLABORATION SYSTEMS AND METHODS

Cross-reference to Related Application

[0001] This application claims the benefit of United States provisional application No. 60/381,060, filed May 14, 2002, which is hereby incorporated by reference herein.

Background of the Invention

[0002] This invention relates to the Internet, and more particularly, to techniques for creating and viewing material on the World Wide Web ("web") in the form of an interactive web book, and to techniques by which multiple individuals can communicate with each other and work collaboratively on content and materials in an interactive web book.

[0003] The World Wide Web has made the Internet accessible to a broad range of people. One can search the web and view large amounts of material using a web browser. Recent improvements have made it easier for individuals to contribute their own ideas and creative content via the web. However, there is no satisfactory framework within the web to assist and enable a group of people who may be physically remote from each other, to talk to each other, work together, discuss ideas, create, view, enhance, modify, alter, comment on, and create multiple alternative versions of new material while recording their interactions and contributions in such a way that participants may be automatically identified and rewarded for their efforts, the deliberations of the group preserved and reviewed, and the sequence of interactions reproduced. As a result, many people who have access to the web do not use it in their collaborative efforts.

[0004] Furthermore, there is no satisfactory framework within the web to support the definition of pre-defined roles and responsibilities for group members, by which groups organize their activities, nor any satisfactory framework within the Web either to assist participants in understanding, reviewing, accepting, or performing those roles, or to implement systems or methods by which those roles may be objectively specified and carried out automatically or semi-automatically by intelligent artificial agents.

[0005] It is therefore an object of the present invention to facilitate the collaboration of multiple participants using a data network such as the web and to record the particulars of their collaboration.

[0006] It is also an object of the present invention to provide a way in which to facilitate a description, specification, and implementation of roles using a data network such as the web.

Summary of the Invention

[0007] These and other objects of the invention are accomplished in accordance with the principles of the present invention by providing a framework and a space referred to as a topic room in which one or more individuals or other entities may collaborate on topics of mutual interest. A topic room is a collaborative framework built upon an interactive web book, in which multiple individuals may carry out live or non-real time written, verbal, or multimedia conversations, and may work together to discuss, create, browse, modify, comment on, disagree about, create enhancements to or alternative versions of, and otherwise contribute to content or material. Within the framework of the topic room, such discussions are recorded and logged, as are all elements of the collaboration, interactions between participants, and individual contributions of those participants.

[0008] A topic room includes a verbol chat room. A chat room holds all of the currently active participants in the topic room, and supports their interactions. The verbol chat room records the activity in the topic room, makes that activity visible to the participants, monitors topic room activity, and carries out various support tasks as requested by the chat room participants.

[0009] Activity in a topic room is a sequence of actions. A record of topic room activity comprises a log of topic room actions, create by the verbol chat room. This log becomes part of the contents of the topic room, and may be selectively inspected, reproduced, and animated. A recorded topic room action can be animated. Animating a recorded action reproduces the action, and animating a sequence of actions has the effect of recreating the activity in the topic room.

[0010] A conversation between participants in a topic room may well include verbal communication. A conventional telephony system may be employed for this conversation. Alternatively, individuals or groups may employ local recording techniques that generate digital audio or other multimedia recordings, and then share those recordings, or portions thereof. The audiorecorder is a topic room tool which supports local or remote capture of audio or multimedia recordings of participants' conversation, singing, recitations, musical or other performances, etc., and assists in the inclusion of such material into the topic room along with useful other information. The other information can include a timestamp, a source identifier (e.g., the speaker), and a topic specification. The audiorecorder can be used to annotate a continuous recording, or to include limited "clips" taken from a larger recording in the topic room record.

[0011] Digital recordings are only one example of material relevant to topic room activity that may not have a native form that allows them to be easily and directly incorporated into an ibook page. Such material is called "external material". Topic rooms include a method for representing external material outside of an ibook folder by a page inside the ibook folder, and for generating an ibook-internal Universal Reference Locator from the ibook-external URL of the external material.

[0012] An itag makes it possible to locate external material. An itag is a structured meta-content object that contains information about ibook folder content. An itag can be embodied as an HTML element. An itag is created for, and stored in, every ibook page. An itag holds all the meta information (non-content information) for an ibook page. An itag holds system-generated information, such as authoring information and various content links. An itag may also contain a vspace object, through which the behavior of the page is defined, using techniques that are discussed below. Finally, the itag of each page may hold a template copy of the istamp for that page.

[0013] Collaborative efforts are aided by techniques which support the tentative and contingent association of comments, notes, alternative renderings, pictures, and other material with the content of ibook pages. Topic rooms support this contingent association with "istamps". Istamps are structured meta-content objects that refer to an ibook page, and can be inserted into one or more ibook pages. An istamp can be embodied as an HTML element. The visual aspect of an istamp can be determined on a case-by-case basis when the page in which it is embedded is displayed. As a result, a particular istamp can be rendered visible or invisible, sets of related istamps displayed in such a way as to share identifying properties such as color, etc. The istamp could also be rendered by rendering the contents of the ibook page to which the istamp refers. The rendering characteristics of an istamp can be determined by the istamp's creator, the owner of the page in which the istamp is embedded, the viewer of the page, or other controlling factors. Istamps can be copied manually or automatically using a "drag-and-drop" or other operation.

[0014] Each participant in the activities that take place in a topic room may be represented by an identity ibook. An identity ibook may be a topic room whose topic is the entity it represents. Whenever an entity participates in any topic room, that entity's activity may also be recorded in their identity ibook.

[0015] All activities in a topic room may be captured and described using a formalism called Verbol. Verbol is an architecture for creating, sustaining, and supporting communication between independent, distributed, self-regulating, cooperating objects known as verbol objects. A Verbol object can do two things: it can communicate with other verbol objects, and it can control non-verbol objects.

[0016] Ibooks folders and ibook pages are verbol objects. Other verbol objects include the audiorecorder, the ibook controller, the chat room, and the ibook server. Verbol objects may communicate with each other by constructing, transmitting, receiving, and interpreting verbol messages. Each verbol message constitutes a plurality of verbol statements. A verbol statement may be a speakable command that can be interpreted by a verbol interpreter and complies with the specifications of a verbol grammar. A verbol statement may define an atomic event or action within a topic room.

[0017] A Verbol script comprises a parallel sequence of verbol statements. Verbol scripts are used to define useful collections of verbol actions, often described as "behavior". All verbol objects have such scripts associated with them. Hence, verbol objects have "behavior" associated with them. One verbol object can inherit the verbol scripts associated with another verbol object, and thereby "inherit" part or all of the "behavior" associated with that other object.

[0018] As described above, every ibook page may have an itag associated with it. In the itag is the vspace for the page. A vspace is the collection of verbol scripts that are attached to the ibook page, and which contribute to the behavior of the page. The behavior of an ibook page is the behavior specified by the page's vspace. This in turn is the behavior of the scripts in that vspace, including any inherited behavior.

[0019] The behavior of an ibook folder is the composition of the behavior of the pages in the ibook folder. Composition is a well-defined inheritance process.

[0020] A persona is an ibook folder whose vspace has been designed to describe useful, interesting, entertaining, or otherwise distinguishing behavior. A participant in a topic room may temporarily or permanently "adopt a persona" by having that participant's identity ibook folder inherit the verbol scripts in the vspace of the persona ibook folder.

[0021] A verbol assistant is a self-contained identity ibook that has sufficiently well-defined, sophisticated, useful, and or interesting behavior, history, and content that it can usefully contribute by itself as a topic room participant.

[0022] Activity within a topic room may constitute parallel sequences of verbol messages exchanged between the verbol objects representing the participants in the room and the material upon which they are collaborating. Records of topic room activity may take the form of verbol scripts, and may constitute the history of their respective topic rooms.

[0023] Further features of the invention, its nature and various advantages will be more apparent from the accompanying drawings and the following detailed description of the preferred embodiments.

Brief Description of the Drawings

[0024] FIG. 1 shows an illustrative ibook site in accordance with the present invention.

[0025] FIG. 2 shows an illustrative topic room in accordance with the present invention.

[0026] FIG. 3 shows a system diagram of an ibook site server, a chat server, and client computers in accordance with the present invention.

[0027] FIG. 4 shows topic room client components in accordance with the present invention.

[0028] FIG. 5 shows a user chat message and verbol chat message in accordance with the present invention.

[0029] FIG. 6 shows an illustrative MySharer window in accordance with the present invention.

[0030] FIG. 7 shows an illustrative chat room window in accordance with the present invention.

[0031] FIG. 8 shows an illustrative audiorecorder window in accordance with the present invention.

Detailed Description of the Preferred Embodiments

[0032] Generally speaking, the present invention provides improvements on existing interactive web book systems. An existing interactive web book system is described in commonly-assigned U.S. Patent 6,052,717 to Reynolds et al (hereinafter "ibook system"), which is hereby incorporated by reference herein in its entirety. An interactive web book system is a self-extending, self-sustaining information-redistributing web robot which is resident on a data network such as the Internet or an intranet. Material in an interactive web book system may be organized in the form of web pages. Material may be text, two-dimensional, three-dimensional or n-dimensional graphics, animation, audio, video, source or executable code, or any type of multimedia format. Within each Web page, material may be organized in passages. Passages can be of any suitable size, such as a paragraph for textual material, or a video or audio clip of a certain length for multimedia material. Related interactive web book system technology supports the automatic distribution, publishing, and synchronization of interactive web book system content and other data between multiple users and sites. These features are described in, for example, Reynolds et al. PCT Publication No. WO 02/088909, filed February 1, 2002 (hereinafter "MySharer"), which is hereby incorporated by reference herein in its entirety.

[0033] <u>Topic Rooms</u>

[0034] A topic room may include all of the behavior and functionality of existing interactive web book systems, such as the ibook system. A topic room also preferably includes a chat room that supports and records interactions between multiple users and verbol scripts that define the behavior and capabilities of the topic room. The term ibook folder while be used herein to refer to material that is stored in an interactive web book system. Therefore, a topic room preferably includes an ibook folder, a chat room, and verbol scripts.

[0035] Every topic room preferably has a topic. The topic is what the material in the ibook folder is about. A subject ibook folder is an ibook folder whose topic is a particular subject (other than being about an ibook member or a persona). An identity ibook folder is an ibook folder whose topic is an individual (or other entity) that is an enrolled member of an ibook site. A persona is an ibook folder whose topic is a history and a named set of behaviors, both of which a contributor can adopt to enhance their identity.

[0036] Ibook Sites

[0037] Topic Rooms are held in ibook sites. ibook sites allow users to gain access to topic rooms by enrolling as members in the ibook site that holds the topic room. Every topic room is preferably contained within a single ibook site. An ibook site may contain multiple topic rooms. An exemplary ibook site is described more detail in the ibook system patent.

[0038] Ibook Folders

As used herein, the terms "ibook folder" and "topic room" are largely interchangeable. Where a distinction appears to be indicated, an ibook folder should be considered to correspond to the aspect of a topic room that has the properties of existing interactive web book systems such as the ibook system. An ibook folder has the capacity to store and present content, maintain attribution and derivation information associated with the content, and so forth.

[0039] Formally, in object-oriented programming terminology, a topic room is an ibook folder: it has all of the properties, methods, and behaviors or an ibook folder, and may well have additional properties methods and behaviors.

[0040] The ibook folder aspect of a topic room stores all of the permanent content of the topic room. That is, the persistent storage of all topic room state information is in the ibook folder of the topic room. The content, history, and behavior of a topic room may be organized into a collection of ibook pages.

[0041] An ibook page comprises an itag and content material. Every ibook page preferably has an itag. The content portion of the ibook page may be empty. An itag holds information about the page, and a vspace. The vspace holds verbol scripts that define behavior associated with the page.

[0042] Each ibook folder preferably has several distinguished pages. One distinguished page may be the "home page" of the ibook folder. The topic of the ibook folder may be specified in the itag of the ibook folder's home page. Other distinguished pages may be "chat pages". A chat page holds a record or log of a verbol chat that took place in the ibook's chat room. Alternatively, a chat page may be located elsewhere within a topic room such as within a verbol chat room.

[0043] <u>Itags</u>

[0044] An itag is a structured text element. Every ibook page preferably has an itag. In one embodiment, an itag may be an HTML element. Every itag preferably contains system-generated identifying information about the ibook page, an archetypal istamp for the ibook page, and the vspace of the ibook page.

[0045] The system-generated information in an itag may include but is not limited to a) template links, also known as attribution links; b) copy links, also known as derivation links; c) identity links, which point to the contents of the ibook page; d) child links, which should be generated in the child list of the vspace of the parent; and e) authoring information, including but not limited to the universal date and time of creation of the content associated with the itag, author or speaker (that is, the creator of the content itself), date of contribution to the ibook; identity of the ibook contributor, and additional information such as type, category, notes, descriptions, etc.

[0046] Vspaces and Verbol Scripts

[0047] The itag in an ibook page preferably holds a vspace. A vspace contains a user-defined verbol script that defines the behavior of the ibook page. Among other things, a page's verbol script may act to automatically incorporate contributions to the ibook page or folder. As a result of the incorporation, some ibook pages may be revised and new ibook pages may be created.

[0048] When an ibook page is created, it normally inherits its behavior, in the form of verbol statements, from its parent and from its template. The parent of an ibook page is the ibook page that the ibook controller of the page's creator was browsing when the page was created. As part of the creation process, the creator can optionally designate any page in the current ibook folder as the template for the page being created.

[0049] The method by which an ibook page inherits verbol behavior may be either static or dynamic. In the static method, when a page is created the contents of the vspace of its parent and its template is copied into the new ibook page's vspace. In the dynamic method, when a page is created the contents of the vspace of its parent and its template is referenced in its vspace. Thus, in the static method, subsequent changes to a page's parent or template behavior does not result in changes in the page's behavior. In the dynamic method, subsequent changes to the parent or template behavior is reflected in the page's own behavior. When the static inheritance method is employed, the inheritance may be "refreshed" from time to time. In the refresh operation, new copies of the parent's and template's behavior are transferred to the page, so that the page acquires any changes in the parent's or template's behavior that took place since the last static copy operation.

[0050] IBook Page Content

[0051] The content associated with an ibook page may reside in the ibook page itself, or external to the page. In one embodiment of ibook pages, content within the page is in the form of HTML statements. Other forms of content include but are not limited to Microsoft Word documents, PDF documents, faxes, spreadsheets, audio and other multimedia recordings, etc.

[0052] External Content

[0053] A topic room uses ibook pages and istamps to incorporate content into an ibook folder. An ibook folder may include any kind of content that can be digitally located and accessed, so that the fundamental advantages of being part of an ibook folder (the ability to access and manipulate the content, the maintenance of a history for the content including at least derivation and attribution information, and the storage and invocation of behavior of the content) can apply to content of any kind.

[0054] Content is incorporated into an ibook folder by creating an ibook page to hold the content, or to refer to the content. All the content associated with an ibook folder is preferably held in ibook pages, or referred to by ibook pages. For example, in one embodiment of istamps, the ibook controller uses Uniform Resource Locators (URLs) to refer to, locate, and access all forms of content. In the embodiment under discussion, the ibook controller also uses a URL to refer to, locate, and access ibook pages. In this embodiment, the ibook controller handles "external content" -- content that can not be inserted into an ibook page, but that is to be logically included as part of the ibook page -- by making use of an algorithm that accepts as input an arbitrary content URL, and produces as output a "mapped" URL that is within the domain of the ibook folder. The ibook controller creates an ibook page with this mapped URL. The itag of the new page includes the URL of the external content. If some content is stored inside of its istamp, it is stored in such a way that it can be easily located once the istamp has been located. Thus either the ibook page or the external content may be located by knowing the URL of the other element.

[0055] <u>Verbol Chat Rooms</u>

[0056] A. <u>Overview</u>

[0057] As discussed above, and as illustrated in FIG. 1, ibook site 100 may include multiple ibook folders 1000. Some of the folders may be identity ibooks folders, and some may be subject ibooks folders. In addition, ibook site 100 may include one or more topic rooms 2000. FIG. 2 shows a more detailed diagram of topic room 2000. As discussed above, and as illustrated in FIG. 2, topic room 2000 comprises an ibook folder 1000 and a verbol chat room 1200. The ibooks folders in topic rooms may typically be subject ibook folders, but may also be identity ibook folders. Ibook folders may include multiple ibook passages such as ibook passages 1100. Topic room 2000 preferably includes a chat page or chat log 1150.

[0058] Verbol chat room 1200 allows ibook site members to exchange messages and contributing content to the ibook folder 1000 within topic room 2000. An interaction of this sort is referred to as "verbol chat," or simply as a "chat". Verbol chat room 1200 (or simply the chat room) supports verbol chats, and generates a complete history of each chat, storing the history in the chat page or chat log 1150 of topic room 2000. The history becomes an integral part of the content of topic room 2000.

[0059] A topic room may include one or more ibook folders and one or more chat rooms. In one possible embodiment, one chat room is created for each ibook folder nominated as a Topic Room. When a Topic Room is created (for instance, during enrollment of an ibook site user, or when an ibook site user issues a "create new ibook folder" command to the ibook site) a chat room is created by the ibook site that is supporting the ibook, and is associated with the Topic Room. Each verbol chat room preferably has a name. Its name may be derived from the name of the Topic Room with which it is associated.

[0060] A verbol chat room may a) allow topic room participants to engage in live, on-line communication with each other via chat messages; b) record their interactions as a log of chat messages in the chat page; and c) is capable of animating those records--playing them back--so as to reproduce recorded chat room interactions. The chat room also reduplicates the contributions of participants to other ibook folders by sending Verbol messages to other relevant ibooks folders. Whenever a participant makes a contribution to a topic room, an entry is added to the log, or history, of that topic room describing the contribution, and a verbol message is also sent to the participant's identity ibook, which results in an entry in the history of that ibook folder.

[0061] B. Server Side Chat Room Components

[0062] An ibook site sponsor maintains a chat server computer on a network in order to support topic rooms. The chat server computer runs a chat server process. The chat server computer may be the same physical computer as the ibook site computer, or they may be different computers attached via a network. In addition to the chat server process, one chat room software component for each active chat room runs on the chat server computer supporting that chat room.

[0063] C. Client Side Software

[0064] When a user enrolls in a topic room, an enrollment program directs the user to download and install any necessary topic room software onto their client computer. In another suitable approach, the software may be downloaded and installed at any later time onto a client computer. The topic room software includes a standard web browser, an ibook controller which creates a Verbol wrapper around the web browser, a verbol chat client (or simply chat client) software, and any other suitable client software for supporting interaction with the topic room.

[0065] D. General Operation

[0066] During operation, verbol chat participants of a topic room using client computers interact with each other by exchanging text, media, and verbol messages, and create, display, and modify ibook pages. As shown in FIG. 3, these interactions are supported by ibook site server computer 2000 and chat server computer 3000. In particular, chat server process 3100 receives chat messages from individual client computers 4000 and 5000 and rebroadcasts those messages to all participants in the chat room of a topic room. Chat server process 3100 also sends all chat messages to the chat room software component running on chat server computer 3000. This chat room software component is responsible for logging all chat messages in the chat page of the associated topic room, and for retrieving those messages on demand for playback to chat room participant. The chat room software component use standard ibook folder software and operations to maintain and access the chat page in which this log, or history, of chat messages is maintained.

[0067] Ibook site server 2000 and chat server 3000 may simultaneously support multiple chat room participants and multiple enrolled non-chat Room ibook site users.

[0068] Ibook site server 2000, chat server 3000, and client computers 4000 and 5000 may be personal computers, laptop computers, mainframe computers, personal digital assistants (PDAs), etc. or any combination of the same. Ibook site server 2000 and chat server 3000 may communicate with client computers 4000 and 5000 over any communications links suitable for communicating data such as network links, dial-up links, wireless links, hard-wired links, etc.

[0069] E. IRC Embodiment

[0070] In one possible embodiment, the Internet Relay Chat (IRC) protocol is used support chat rooms. In such an embodiment, the chat server is built on top of a standard IRC server, and chat messages are text messages wrapped inside the IRC message protocol. In this embodiment, the verbol chat client communicates with the chat server over a network or Internet using the IRC protocol.

[0071] F. Chat Room Activation

[0072] An enrolled ibook site member needs to log into a topic room before he or she can participate in a verbol chat. In addition to logging in, the member also needs to "enter" the chat room of the topic room. Once they have entered the chat room, they become a participant in the verbol chat. Users may enter a chat room by running their verbol chat client software on their client computer, and providing certain information to the chat client, such as the name of the chat room they want to enter, and the URL of the chat server that is supporting that chat room. Alternatively, this process may be automated so that a user may simply select a chat room icon in the topic room to enter the chat room.

[0073] When an enrolled user enters a chat room, the chat server determines whether or not the chat room is active. If the chat room is not active, the chat server accesses the topic room database, activates the chat room, initializes the chat room software component running on the chat server computer, and enters the user as a participant in the chat room. The chat room will remain active as long as it has any participants. When it no longer has any participants, the chat server preferably deactivates the chat room.

[0074] FIG. 7 shows an illustrative chat room window 8000. As illustrated in the top portion of window 8000, there are two participants in the chat room: gerald and steve2. The top portion of window 8000 also shows their most recent action, which is "browses" for both participants and the locations of their web browser.

[0075] G. Chat Room Participation

[0076] Chat room participants can send and receive messages to and from other participants, and make various requests of the chat room and of other participants. For instance, they can ask the chat room to let them "follow" another participant as the participant browses from place to place. Participants make such requests using the verbol chat client. The chat client may perform at least the following functions:

[0077] 1. Display relevant information, including the names, actions, and locations of participants, and certain selected chat messages.

[0078] 2. Provide a user interface for issuing chat room requests.

[0079] 3. Filter chat messages, selecting certain messages for display.

[0080] 4. Hide chat messages, so that they are not displayed.

[0081] 5. Monitor chat messages, and execute predefined processes when certain messages, or combinations of messages, are detected.

[0082] The requests that a chat client supports include the following:

[0083] 1. Enter a designated chat room.

[0084] 2. Leave the current chat room.

[0085] 3. Follow another participant in the chat. A participant may follow another participant for example by right clicking on the other participant in window 8000 and selecting this option.

[0086] 4. Browse to the same location as another participant in the chat. A participant may browse to the same location as another participant by selecting that participant's location in the top portion of window 8000.

[0087] 5. Send a user message to the chat room. A participant may send a message to the chat room by entering the message in message text box 8050 in window 8000.

[0088] 6. Show older messages in the chat room. A participant may view older messages by selecting menu option 8100 in window 8000 and selecting this option.

[0089] The chat client supports these requests by gathering necessary information from the participant, and then creating and transmitting chat messages. The chat client also receives all chat messages broadcast by the chat server, examines each messages, and processes it appropriately. Depending upon certain predefined participant preferences, the chat client may display some of the messages it receives. Chat messages are displayed for example in area 8150 of window 8000. The chat client also passes all chat messages to other client topic room software components. When a topic room user is participating in a chat room, all of the topic room participants' client software components monitor all chat messages.

[0090] H. Chat Messages

[0091] Topic room software generates two types of chat messages: user chat messages and verbol chat messages. User chat messages are generated by the chat client whenever the user types, or otherwise creates, a message to chat room participants. User chat messages may have arbitrary message content that is examined but often not interpreted by topic room software components. Verbol chat messages are chat messages that are generated by topic room client or server software components. Verbol chat messages contain verbol commands that are typically interpreted by and acted upon by other topic room software components. Thus, in general, both human participants and software clients create, monitor, examine, and react to chat messages.

[0092] FIG. 5 illustrates content 6300 and 6600 of user chat message 6100 and verbol chat message 6500, respectively. Content 6300 of user chat message 6100 comprises the following fields: a) a user name indicating the name of the participant who generated the message; b) a timestamp indicating when the message was generated, and c) an arbitrary message content that the user entered. Content 6600 of verbol chat message 6500 also comprise the user name field and the time stamp field. In addition, verbol chat message 6500 also comprises a designated prefix field (the string "VERBOL!" in one possible embodiment), a recognized verbol command field (such as "BROWSE", "REVISE", or "CREATE"), and a set of command arguments (such as a URL).

In one possible embodiment cat messages are "wrapped" in an IRC protocol for transmission between software components on a client computer, for transmission from a client computer to the chat server, and for transmission from the chat server to all participant client computers.

[0093] Topic Room Client Components

[0094] A topic room client computer runs software designed to support ibook folder contribution and chat room participation. This software may be downloaded (e.g., from an ibook site) during topic room user enrollment, or at a convenient later time. As shown in FIG. 4, a client computer 5000 runs a web browser 5200 to support ibook folder contribution, and a verbol chat client 5400 to support chat room participation. The client computer may also be running additional ibook client components. All of these components communicate with each other by exchanging verbol chat messages through a communications component referred to as "Coms Layer" 5500. Chat client 5400 receives all chat messages from the chat room (e.g., chat room 3200 (FIG. 3)), interprets each message, and also passes it on to the other client components by sending the message to the Coms Layer 5500.

[0095] The web browser used to access the ibook folder is customized by wrapping it in an ibook controller component 5100 that supports specialized ibook folder operations. This controller also acts as a verbol wrapper, translating the web browsers actions into verbol chat messages, and allowing other components to interact with the web browser by sending it verbol chat messages. The web browser continues to communicate with the ibook web server using, for example, standard web messages, such as HTTP messages. In addition, whenever a topic room user is participating in a verbol chat room, every action the ibook controller takes, such as creating a new ibook page, browsing to a different ibook page, or revising an ibook page, is expressed as a verbol command that is transmitted as a verbol chat message to the chat room. As a result, the log of the chat room contains a complete history of the action of all participants in the chat room. In addition, any participant "A" can browse to the current location in the topic room that any other participant "B" is at. Furthermore, any participant "A" can "follow" any other participant "B" from location to location in the ibook folder. The "follow" operation would be performed as follows:

[0096] 1. Participant A selects participant B from the list of current chat room participants displayed by participant A's chat client;

[0097] 2. Participant A indicates that participant B should be "followed" by selecting a "Follow" control (such as a menu option) displayed by participant A's chat client;

[0098] 3. Participant A's chat client issues a verbol chat message specifying the follow action, and the name of the participant to be followed, and transmits that message to the chat server and to the Coms layer of the client computer.

[0099] 4. Participant A's ibook controller has registered with the Coms layer, and thus receives the follow verbol chat message. It registers the fact that from now on it is to monitor all chat messages for "BROWSE" verbol chat messages from participant B, and when it finds such a message it is to instruct the web browser that it is wrapping to browse to the URL specified in participant B's chat message.

[0100] 5. Participant A's "follow" chat message is also sent to the chat server. The chat server sends it to the chat room software component running on the chat server computer, and to all of the chat room participants.

[0101] 6. The chat room server component receives the chat message and writes it to the chat page of the Topic Room.

[0102] If at some later time Participant A wants to stop following Participant B, the following steps are taken:

[0103] 1. Participant A selects a "stop following participant B" control on his chat client.

[0104] 2. Participant A's chat client issues a verbol chat message specifying the "stop following" action, and transmits that message to the chat server and to the Coms layer of the client computer.

[0105] 3. Participant A's ibook controller receives the stop following verbol chat message. It registers the fact that from now on it is to ignore any chat messages for "BROWSE" verbol chat messages from participant B, and does so.

[0106] 4. Participant A's stop following chat message is also sent to the chat server. The chat server sends it to the chat room software component running on the chat server computer, and to all of the chat room participants.

[0107] 5. The chat room server component receives the chat message and writes it to the chat page of the topic room.

[0108] <u>History Activation</u>

[0109] Playing back the history of participation in a chat room is another important capability. Because ibook folder content is not removed or modified, only revised (that is, a new copy made with the desired modifications), any previous ibook folder state can always be redisplayed. For instance, a participant can ask for the last 10 minutes of chat room activity to be re-enacted, as follows:

[0110] 1. Participant A selects "replay recent history" from a control on his or her chat client.

[0111] 2. The chat client asks participant A how many minutes back the playback should begin. Participant A enters a response (for example "10 minutes").

[0112] 3. Participant A's chat client issues a verbol chat message specifying the replay history action, and the time value of 10 minutes, and transmits that message to the Coms layer of the client computer and to the chat client.

[0113] 4. Participant A's ibook controller has registered with the Coms layer, and thus receives the replay history verbol chat message. It registers the fact that from now on it is to monitor all chat messages for "BROWSE" verbol chat messages with its own name, and when it finds such a message it is to instruct the web browser that it is wrapping to browse to the URL specified in those chat messages.

[0114] 5. Participant A's replay history chat message is also sent to the chat server. The chat server sends it to the chat room software component running on the chat server computer, and to all of the chat room participants.

[0115] 6. The chat room server component receives the replay history chat message and interprets the message. It acts on the message by retrieving all of the messages it has received during the last ten minutes and resending them to the chat server in chronological order.

[0116] 7. The chat server broadcasts all of the recorded messages.

[0117] 8. The recorded messages are received by participant A. As participant A's chat client receives each recorded message, it displays the message. As participant A's ibook controller component receives the messages, it filters them and directs the web browser it controls to browse to the location that participant A had browsed to during the designated times.

[0118] <u>Verbol "Whenever" Statements</u>

[0119] Chat client software components monitor chat messages by acting as chat listeners. A chat listener is an object that monitors chat messages, and detects certain designated messages or combinations of messages arriving in a chat room. A client component becomes a chat listener by registering with the Coms layer. In one possible embodiment, it does this by calling the Coms layer with the address of a callback function to be invoked whenever the Coms layer receives a chat message.

[0120] Each client component may handle every chat message, but a particular component may respond to many chat messages by ignoring them. However, when certain designated chat messages, or combination of chat messages, are detected, the chat listener may automatically trigger certain scripted processes. In one possible embodiment, each triggering chat message or chat message combination, and the scripted process to execute in response, is specified using a verbol "whenever" statement stored in the vspace attached to or associated with the current ibook page. The verbol whenever statement has the following form: whenever <event pattern> <action pattern>. The event pattern may be any regular expression defining one or more users and their actions as logged in verbol statements in the chat room, and action pattern may be any action or process expressed in a scripting or other language such as verbol, Java, JavaScript, Perl, PHP, etc. For example, the following is a verbol whenever statement: "whenever {any page/this page is} revise[d] email [it to] email-address-list," where the text in square brackets is optional, and does not change the meaning of the message. The text in braces are alternate forms meaning, respectively, any web page or the current web page, and "email-address-list" is a valid Internet electronic mail address list.

[0121] <u>Ibook Contributors and Chat Room Participants</u>

[0122] Topic rooms are used by members of ibook sites. Ibook site members can contribute to ibooks folders on the site, and participate in the site's chat rooms. In order to contribute or participate, each user may be required to enroll as an ibook site member (a one-time operation) and then log into each topic room to which they want to contribute. The member's contribution may take the form of participation in the chat room, or contribution of material to the ibook folder, or both. Making a contribution to the ibook folder includes participation in the chat room, because the topic room records a complete history of all contributions as entries in the chat room record.

[0123] The terms "contributor" and "participant" are largely interchangeable. The term "contributor" focuses on people who contribute content to the ibook folder. The term "participant" focuses on people who participate in the chat room. A person can be a participant without being a contributor (if you only exchange messages with other participants, but never make a contribution to the ibook), but if you are a contributor you are also a participant, because every ibook contribution results in an entry in the chat room history.

[0124] The topic room may be considered a participant in its own chat room. When it updates the chat history files in its ibook folder, it creates a message in the chat history reflecting that contribution. Therefore it is listed as a participant, and can be found in the list of chat room participants.

[0125] Thus, some participants are human, other participants may be legal entities, and still others are objects such as topic rooms.

[0126] Identity Ibooks Folders

[0127] Every enrolled member of an ibook site has an identity ibook folder on that site. An identity ibooks may be a part of a topic room whose topic is an ibook site member. The site member may be a human individual or it may be a legal entity such as an organization or corporation.

[0128] Other identity ibooks folders describe purely verbol constructs -collections of content, behavior, and capabilities. This sort of identity ibook is called a verbol assistant. Verbol assistants can contribute to ibooks, and participate in chat rooms. For example, a meeting coordinator assistant is a set of verbol scripts, stored in an identity ibook folder, that knows how to coordinate meetings of remote participants. It includes scripts for calling individuals, connecting them to the conference call, etc. A glossarizer is another example of a verbol assistant that makes contributions to an ibook folder by monitoring the contents of the ibook folder, finding important words and phrases, linking them to glossary pages in the ibook folder, and helping (human) contributors add new glossary pages to the ibook folder.

[0129] The history of an identity ibook folder may include a complete record of all contributions that its entity has made to ibooks folders on the site.

[0130] Persona

[0131] A persona is an ibook folder that defines and names a collection of behaviors and capabilities that have been designed to be adopted -- that is, inherited by -- an identity ibook folder. Ibook site members may temporarily take on a persona by having their identity ibooks folders temporarily inherit the verbol scripts of a persona ibook folder. Some personae are intended to be adopted by human entities and others are intended to be adopted by verbol assistants.

[0132] Roles

[0133] A role is a specification of what is to be achieved by the behavior of a persona. Some roles are intended to be adopted by human contributors and others are intended to be adopted by verbol assistants.

[0134] Istamps

[0135] An istamp is a named, addressable, structured text object that points to some content. In one embodiment, an istamp is an HTML element. An istamp is designed to be used in ibook folder content to "point to" other ibook folder content, such as an ibook page. An istamp's name and address is intended to make it easy to link to istamps from within other ibook locations. Every itag includes an istamp for the ibook page containing the itag. An istamp is designed to be inserted into ibook content. Tools that manipulate istamps generally support a "drag and drop" operation. Therefore users can drag a page's istamp to a new location, and copy that istamp by "dropping" it into that location.

[0136] An istamp contains a reference to some content. In one embodiment, this reference takes the form of an HTML anchor element. The value of the "href" property of the element points to the content that the istamp designates.

[0137] Display of Itags and Istamps

[0138] Itags and istamps can be displayed when their associated ibook page is displayed, or they can be hidden. Any ibook contributor who creates an itag or an istamp can indicate whether that element is to be displayed or to be hidden. The viewer of an ibook page can also indicate whether they want to have itags and istamps displayed, hidden, or handled according to the specification of the element's creator. The preference of the viewer may take precedence over the preference of the element's creator.

[0139] Verbol Chat Generally

[0140] In one embodiment, the chat room of a topic room is implemented using verbol chat. In this embodiment, chat room participants chat by exchanging verbol messages. Verbol chat supports collaborative conversations by the participants in the topic room.

In one embodiment of verbol chat, participants use the IRC protocol to exchange messages. Text messages can be typed by each participant, including verbol assistant objects. The messages appear in a chronological, scrollable log in a "verbol chat" window on the screen of all of the participants. Each entry in the chat log corresponds to a message from one of the participants, or one of the verbol assistants. Each message is tagged so as to identify its origin. It is also time stamped.

[0141] A verbol chat room allows every object to receive messages from other objects in the chat room. Although users chat to each other in unrestricted text, or send pictures, or send messages with similarly unrestricted contents, the process of sending a message results in a verbol command being generated. This verbol command "wraps" the message contents, and is recorded in the history of the verbol chat by the verbol chat controller.

[0142] In addition to sending comments, topic room participants can tie their browser displays to the display of another participant, in effect "following" that "leader" around as she or he browses content. In this mode, the leader's ibook controller generates verbol messages that specify where the "leader" is browsing. These verbol messages are sent to the chat room. Any other participants you have become followers can follow the leader by having the topic room send the leader's verbol messages to each follower's ibook controller. When a follower's ibook controller receives such a verbol message, it will direct the follower's browser to browse to the same location as the leader.

[0143] Participants in chat rooms have, or can have, roles (see Roles above).

[0144] Fundamentally, verbol chat generally comprises: a) participants who use verbol to participate in the chat room and make contributions to the ibook; b) a history log of all the verbol commands for those interactions and contributions; c) a mechanism for capturing and displaying those verbol scripts; d) a mechanism for listing the participants; e) a mechanism for storing, searching, and locating those histories, and episodes within those histories; and f) a mechanism for animating the verbol histories (that is, a means of executing the logged verbol commands so as to recreate the history of the verbol conversation and contributions).

[0145] Searching Verbol Histories

[0146] Verbol chat logs can be scanned and filtered based on the properties of the verbol commands in the logs. Some of the properties that can be combined to search or filter the logs include the name of the topic room, the name of the participants in a chat, the temporal characteristics of the script elements (time, date, duration, etc.), the types of roles played by the participants, the nature and effect of the verbol commands that comprise the logs, and location properties of the content and messages contributed in the log.

[0147] Voice Chat

[0148] As discussed above, in order to log into an ibook site, an ibook site member may be required to launch an ibook controller. The version of ibook controller used with topic rooms may include an "audiorecorder" controller or audiorecorder. The audiorecorder helps contributors record, markup, save, and playback audio material, including material spoken by the user. The audiorecorder is software that controls standard audio, multimedia, and/or telephony hardware on the contributor's computer.

[0149] Using the audiorecorder, audio content can be added to an ibook folder. This audio content is added as "external content" as described above. Such a page is called an "audiopage".

[0150] More particularly, recorded audio content can be downloaded to the server that is hosting an ibook site. An ibook page can be created whose itag points to the external audio content (this is the audio page). Using the audiorecorder, istamps that designate specific locations and/or regions in the audio recording can be added to the audiopage. These istamps are called "index points". When a user clicks on an index point in an audiopage, the audio content beginning at the designated location, or within the designated region, will be played back for the user.

[0151] In one embodiment, the "MySharer" technology is used to transfer audio content from a participant's machine to the ibook site. In this embodiment, each participant specifies through MySharer that they want a particular file folder on their computer to be regularly updated and synchronized with a corresponding folder on the ibook site. Each participant may specify that their audiorecorder is to drop audio files into their local copy of the shared folder. When a person makes a recorded comment in the topic room, an audio file with a recording of their comment is placed in their local copy of the shared folder. MySharer may then copy that file into the ibook site synchronized folder. From there, MySharer may propagate a copy of the recorded comment to the corresponding folder on each participant's computer.

[0152] Software Usage

[0153] The following explains how to integrate one specific embodiment of a standalone audiorecorder, MySharer, and chat, to record, send, and play back voice messages while communicating with people in a chat room. When a topic room member creates a recording, it will be created in a directory selected by the member. This directory is specified as part of a MySharer mapping. MySharer will therefore upload the recording file to a selected ibook folder. When MySharer has uploaded the file, a message may be displayed in the ibook chat window. This message may be selected by the member to link to the file in the chat window, and the file will play.

[0154] To use voice chat:

[0155] 1. Create a MySharer mapping - This enables MySharer to upload the wave file of your recording to an ibook site (e.g., free.familysystems.org).

[0156] 2. Start ibook chat - This enables the user to see that MySharer has carried out the upload request, and to click on a link to the wave file to play it back.

[0157] 3. Create a recording using audiorecorder - Record your voice input, using for example an external mike attached to your PC's sound card.

[0158] A user needs to create the mapping between the directory on his or her client computer in which the audio recordings are saved, and a MySharer directory on an ibook site (e.g., free.familysystems.org) If an appropriate mapping already exists, then this is not necessary.

[0159] To create a mapping, a user may:

[0160] 1. Start MySharer (e.g., build 138, which is part of build 144 of the ibook controller).

[0161] 2. Click MySharer/Options - the MySharer options dialog is displayed.

[0162] 3. Click the Add to create a new mapping. A "Nominated Dir Properties" window may be displayed. FIG. 6 shows an illustrative "Nominated Dir Properties" window 7000.

[0163] 4. In the field local directory, enter the name of directory from which the voice messages will be uploaded. This may be peformed by using the browse button to select the directory, or if a standalone audiorecorder is running, highlight the path (such as C:\My Recordings) in the recordings directory field in the Audio Recorder window, and press the keys Ctrl + C to copy this value to the Windows clipboard, then press Ctrl + V to paste it into the Local Directory field.

[0164] 5. In the field Web Address, enter the full web address of his or her ibook folder. The user should make sure, for example, that he or she has enrolled on free.familysystems.org - the user should then have an identity ibook folder in their name. The user may open his or her ibook folder, and click the MySharer ibook folder link in his or her identity ibook folder. The URL should now look like this: http://free.familysystems.org/identity-ibook/myname/mysharer/myfilelist.htm. The user may copy the path of this URL (excluding the filename myfilelist.htm) and paste it into the Web Address field in the "Nominated Dir Properties" window. For example, copy: http://free.familysystems.org/identity-ibook/myname/mysharer.

[0165] 6. Enter his or her user ID and password for his or her identity ibook folder in the Userid and Password fields.

[0166] 7. Click apply to apply your changes.

[0167] 8. Click the general tab and make sure that the send chat messages check box is selected. Set update, for example, to one minute.

[0168] 9. Click OK to save the changes to the general tabset and exit from the MySharer options window.

[0169] 10. To enable MySharer to automatically upload his or her files from the nominated directory at the specified interval, click start on the MySharer window.

[0170] To start ibook chat, a user may:

[0171] 1. Start the ibook controller, go to the chat menu, and connect to the chat server (e.g., the free.familysystems.org server, and the chat room is talk). The chat room is displayed. FIG. 7 shows an illustrative chat room window 8000.

[0172] 2. Make sure that verbol commands are shown. (To check if verbol commands are shown, click menu and then options on the chat room window. The options dialog is displayed. Make sure the show Verbol commands check box is checked.)

[0173] 3. Each time MySharer uploads the voice message files, see, in the chat room window, a "MySharer has updated" message against the name of a user. Below it, there will be a link to the voice message. The user may click the link to play back the voice message.

[0174] To start a recording, a user may:

[0175] 1. Record his or her voice message. This may be done with an external microphone connected to the client computer's sound card.

[0176] 2. Start the standalone audiorecorder (e.g., build 27) by for example clicking Start > Programs > Audio Recorder V.1.0.27 > Standalone Audio Recorder. FIG.8 shows an illustrative audiorecorder window 9000.

[0177] 3. Choose the directory where he or she wants to store the voice messages. The default illustrated in FIG. 8 is C:\My Recordings, but any suitable directory may be selected.

[0178] 4. Using his or her microphone to record your message, click the record button to start recording and click stop to finish it. The recording will be saved in the chosen directory and MySharer will copy the file to the chosen ibook folder, and send a message to ibook chat.

[0179] <u>Topic Room Properties</u>

[0180] Every topic room page may have its own set of rights and privileges, which may be determined by some of the statements in its verbol script. Every ibook site member may have his or her own rights and privileges, which are specified in that member's identity ibook. Many roles carry with them certain rights, privileges, and constraints, which are specified in the role ibook.

[0181] The final rights, privileges, and constraints of a participant in a chat room are determined by a well-defined composition of the rights, privileges, and constraints specified in the topic room folder, the participant's identity ibook folder, and the participant's current Persona, if any.

[0182] Verbol

[0183] A. Behavior of Ibook Objects

[0184] The behavior of an ibook folder is the composition of the behavior of that ibook folder's pages. The behavior of an ibook page is defined by the verbol script in the vspace of the page's itag.

[0185] B. The Verbol Architecture

[0186] Verbol is an architecture for creating, sustaining, and supporting communication between independent, distributed, self-regulating, cooperating objects. An object designed to exhibit some interesting or useful behavior within the verbol architecture is called a verbol object. Verbol objects exhibit their behavior when they are activated within a virtual computing environment called the verbol minimum operating environment. The behavior of verbol objects is specified in verbol scripts associated with those objects. Verbol objects have two types of behavior: they exchange messages with other verbol objects, and they control non-verbol objects.

[0187] Within the verbol minimum operating environment, verbol objects may communicate with each other using any language they like. Obviously, objects that don't understand the same language can't communicate effectively, so useful interaction between verbol objects only occurs when the objects that are trying to communicate employ the same language. Colloquially, it may be said that verbol objects "speak verbol," but two perfectly legitimate verbol objects may understand completely different languages, and be utterly unable to communicate. On the other hand, the behavior of a single verbol object, as found in the verbol script for that object, could be specified in a language that only that object understood.

[0188] A verbol object may use a verbol interpreter to analyze and process its scripts and any verbol messages it receives. The verbol interpreter, in turn, may make use of a verbol grammar to parse the messages and scripts, and a verbol processor to process the parsed messages and scripts. The language that a verbol object understands may be completely determined by the verbol interpreter and the verbol grammar that it uses to analyze and process its scripts and messages.

[0189] Therefore, two verbol objects can communicate reliably if and only if they use the same, or compatible, verbol interpreters and grammars.

[0190] Topic rooms, ibook folders and pages, istamps, the ibook controller, and the verbol chat controller are all verbol objects. They are called verbol ibook objects.

[0191] The ibook controller may contain a verbol minimum operating environment. The ibook controller places each verbol ibook object it holds into its verbol minimum operating environment, thereby activating the object and endowing it with the behavior specified by its script.

[0192] Verbol ibook objects may all speak a language referred to as the verbol standard ibook language. Verbol ibook objects may use the verbol standard ibook language both for the messages they send and receive, and for the scripts that define their behavior.

[0193] Verbol ibook objects may all speak the same language because they all use the same verbol interpreter and grammar. The verbol interpreter they use is called the verbol standard ibook interpreter, and the grammar is called the verbol standard ibook grammar.

[0194] C. Verbol Architecture Design Principles

[0195] Verbol objects may control all their own behavior (including all copies). Verbol objects may inherit all contributed behavior. Verbol objects are called by their name. Each Verbol object may be its own class. A Verbol wrapper can be created for any non-Verbol object.

[0196] D. The Verbol Minimum Operating Environment

[0197] Every object may inherit the ability to interpret verbol. The verbol minimum operating environment may be responsible for a) creating, maintaining, running, and terminating verbol objects; b) passing messages between verbol objects; and c) mapping verbol names to specific verbol objects, and for locating those objects.

[0198] The verbol minimum operating environment is a non-verbol object. Like all non-verbol objects, it can be given a verbol wrapper. One embodiment of the verbol minimum operating environment for ibook folders is implemented in Java.

[0199] E. The Verbol Standard Ibook Language

[0200] There are two types of statements in the verbol standard ibook language: "listening statements" and "action statements" (sometimes referred to as "imperative statements"). There are two common listening statements: the "whenever" statement and the "for" statement.

[0201] The whenever statement, as discussed above, has the following form: Whenever <event pattern>

<action pattern>. When an ibook page is loaded, the ibook controller may execute all of the whenever statements in the page's vspace. Each whenever statement represents a condition for which the controller is to "listen", and an action to be taken if that condition is found. The actions are verbol action statements. When an action is to be taken, the corresponding verbol action statement is executed.

[0202] The for statement has the following form: For <action pattern> <vspace designator>. Whenever a verbol object receives a command (a verbol action statement), it may search for either a whenever statement whose event pattern matches the verbol action, or a for statement whose action pattern matches the verbol action. If a matching for statement is found, the verbol script in the designated vspace is scanned to "resolve" the action to be taken to execute the action statement being processed. The vspace designator may be an expression that typically resolves to another ibook page.

[0203] Commonly used verbal action statements may include for example: a) copy [in/to] <ibook> / own book; b) play <audiopage istamp>; and c) create <new page name> [based on] <template specifier>.

[0204] The foregoing is merely illustrative of the principles of this invention and various modifications can be made by those skilled in the art without departing from the scope and spirit of the invention.

[0205] Source Code and Build Code

[0206] This section contains source code and build instructions for the Chat extensions to the Family Systems' ibook controller.

This source code represents partial embodiments of the methods described above. Family Systems Ltd. products and licenses are available at www.ibook.com.

[0207] Build Instructions:

[0208] 1. Extract the source code listed below into java source files, and store the java files in a directory tree. Each source code module is identified with a standard format comment, which includes the expected relative directory into which the file should be extracted in order to construct the necessary Java package structure. Each java source file should be given the extension ".java". Example: the first java class contains the java package statement "package com.fs.chatext;" Therefore, it should be extracted into a file with the name "chatext.java", in a directory with a relative path name of "./com/fs."

[0209] 2. Download the ibook controller from www.ibook.com.

[0210] 3. Install the ibook controller.

[0211] 4. Set the CLASSPATH environment variable to include the directory into which the ibook controller is installed.

[0212] 5. Compile the java source files created in step 1, using the classpath as modified in Step 3.

[0213] 6. The compiled classes need to be available on the classpath when starting the ibook controller - there are various methods for doing this. The ibook controller tries to load a class with the name "ChatExtension" in the default package. If it succeeds, Chat functionality is added to the features available to the user of the ibook.

[0214] <u>Source Code</u>

```
//Source file:
./chat/ChatExtension/src/ChatExtension.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++
//package com.fs.chatext;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */
import java.util.EventObject;
import com.fs.chatclient.*;
import com.fs.chatextension.*;
import javax.swing.*;
import java.net.URL;
import java.net.MalformedURLException;
import java.awt.Rectangle;
import java.util.*;

public class ChatExtension implements
IbookExtensionInterface
{
    static private ChatExtension g_theInstance = null;
// the one and only instance

    private ibookedit       m_ibookedit ;

    // configuration settings
    public String   m_strChatServer;
    public String   m_strChatNick;
```

```
        public String   m_strChatChannel;
        public int      m_nChatPort;
        public String   m_strChatPassword;
        public String   m_strChatUser;
        public String   m_strChatRealName;
        public Rectangle m_rectIbook = new Rectangle();
    // rectangle to position ibook
        public String   m_strChatDefServer;
    // default server for ibook chat rooms
        public boolean  m_bShowVerbol;
    // whether we should show verbol in the chat room
        public Hashtable m_hashTrustedVerbolUsers = new
    Hashtable();          // who to accept Verbol from ?

        // for tracking the Ibook embedded chat
    functionality
        //private String          m_strIbookPassword = "";
    // current password for user logon (captured from
    LoginEvent)
        private String          m_strIbookCurrentHost = "";
    // current Ibook Host (server)
        private String          m_strIbookCurrentIbook =
    "";     // current Ibook location (used to map to chat
    room)
        private ChatInstance    m_ibookchatui = null;
    // instance of UI to use for ibook chat
        private ChatModel       m_ibookchatmodel = null;
    // current chat room

        private Vector          m_vecInstances = new
    Vector();   // chat instances opened by this extension
    private ChatServerResolver m_resolveServer = new
ChatServerResolver();  // helper class

    private DirtyComms      m_dirtycomms = null;  //
quick and dirty comms for MySharer

/*****************************************************
     * Main method primarily for testing
     */
    static public void main(String args[])
    {
        preloadClasses();
        ChatExtension ext = new ChatExtension(null);
// so there is something to set options on  (test only
!)

        ConnectionDialog dlg = new ConnectionDialog();
        ServerConnectionInfo info = new
ServerConnectionInfo();
        info.m_strServer = "free.familysystems.org";
        info.m_strUser = "";
        info.m_strRealName = "John Smith";
        info.m_strNick = "smitty";
        dlg.setConnectionInfo(info);
        dlg.setChannel("talk");
        dlg.setModal(true);          // use modal when
running standalone.
        dlg.setVisible(true);

        if (!dlg.wasConnectPressed())
            System.exit(0);          // kill all threads
    }
```

```
/*******************************************************
     * Returns true if we're running outside of the
ibook.
     */
    public boolean isRunningStandalone()
    {
        return (m_ibookedit == null);
    }
/*******************************************************
     * Returns the location of the ibook at present (or
empty string if none)
     */
    public String getIbookUrl()
    {
        if (m_ibookedit != null)
            return m_ibookedit.currentUrl;      //
fortuanately this was package private
        else
            return "";
    }
/*******************************************************
     * Returns the current ibook password.
     */
    public String getIbookPassword()
    {
        if (m_ibookedit != null)
            return m_ibookedit.getCurrentPassword();
        else
            return "";
    }

    void addInstance(ChatInstance obj)
```

```
        {
            m_vecInstances.addElement(obj);
        }

        void removeInstance(ChatInstance obj)
        {
            m_vecInstances.removeElement(obj);
        }

/******************************************************
       * Returns the ONLY instance of this extension
*******************************************************
        static public ChatExtension getInstance()
        {
            return g_theInstance;
        }

/******************************************************
       * This constructor is public and is ONLY called by
the ibook controller
       * to instantiate an instance of the extension. It
is called ONCE when the
       * ibook starts.

******************************************************/
        public ChatExtension(ibookedit ref)
        {
            g_theInstance = this;           // record the
one and only instance
            m_ibookedit = ref;              // save the ref
for other function calls
            System.err.println("ChatExtension instantiated
- ref=" + ref);
```

```
            m_dirtycomms = new DirtyComms(this);
            m_dirtycomms.start();

            preloadClasses();
        }
    /*****************************************************
         * Ibook callback
         */
        public void onEvent(EventObject evt)
        {
            try
            {
                System.err.println("**
    ChatExtension.onEvent(" + evt.getClass().getName() + ")
    " + evt.toString() );

                if (evt instanceof browserObj)
                {
                    browserObj browseobj = (browserObj)evt;
                    int nID = browseobj.getID();
                    switch (nID)
                    {
                        case browserObj.NAVIGATECOMPLETE:
                        {
                            // tell all the models to
    BROWSE to a new location...

    notifyModelsOfNewLocation(ChatVerbol.BROWSES_VERB,
    browseobj.getURLStr());
                        }
                        case browserObj.DOCUMENTCOMPLETE:
                        {
                                //System.err.println("CHAT:
    Document Complete To: " + browseobj.getURLStr());
                                handleNavigateComplete();
    //browseobj.getURLStr()); - ignore URL - use ibooks
    internal variables
                            }
                        }
                    }
                    else if (evt instanceof IbookEditEvent)
                    {

    handleIBookEditEvent((IbookEditEvent)evt);
                    }
                    else if (evt instanceof URLPostEvent)
                    {
                        handleURLPostEvent((URLPostEvent)evt);
                    }
                    else if (evt instanceof LoginEvent)
                    {
                        handleLoginEvent((LoginEvent)evt);
    // called to check if ibook has changed (typically,
    we've logged in)
                    }
                }
                catch (Throwable e)
                {
                    // attempt to catch all exceptions as
    Ibookcontroller isn't too good at handling the fallout
                    System.err.println("** ChatExtension
    Exception " + e);
                    e.printStackTrace();
                }
            }
```

```
/*****************************************************
      * Returns the server name and port for the
specified ibook host.
      * Currently just returns a hard-coded string. In
future will need to ensure
      * that either the ibooks are rolled out on
different servers, or can be
      * mapped, e.g. using a fixed text file on the
server concerned.
      */
     private ServerConnectionInfo
getChatServerForIbook(String strHost, String strIbook,
String strUser, String strPass)
     {
          ServerConnectionInfo info =
m_resolveServer.getServer(strHost, strIbook, strUser,
strPass);
          log("Connection for " + strHost + " " +
strIbook + " = " + info);
          // work off a copy of the information (just in
case it gets modified - shouldn't happen)
          if (info != null)
               info = (ServerConnectionInfo)info.clone();
// work on a copy for now
          return info;
     }

/*****************************************************
      * Returns the name of the channel to use for a
specific ibook.
      * Currently this is fairly arbitrary based on the
name of the ibook
```

```
     * Need to do something more sophisticated as this
makes the channel name
     * very long, which can impact the max length of
message we might be able to
     * send to some servers
     */
    private String getChannelForIbook(String strHost,
String strIbook)
    {
        // replace spaces with underscores.
        // replace "/" or "\" with period.
        strIbook = strHost + strIbook;
        strIbook = strIbook.replace(' ', '_'); //
spaces are the real problem

        // we now let the server construct a safe name
to use
        return strIbook;
    }

/*****************************************************
     * handling an attempt to update or revise the
current page
     */
    private void handleURLPostEvent(URLPostEvent evt)
    {
        // TODO - need to check the event for "CLOSING"
the ibook window
        switch (evt.getID())
        {
            case URLPostEvent.REVISIONINFO:
```

```
notifyModelsOfNewLocation(ChatVerbol.REVISING_VERB,
m_ibookedit.currentUrl);
                    break;

                case URLPostEvent.POST:
                    // detect the difference between the
POST when the Editor closes, and a POST when
                    // the Editor doesn't close. URLPost
function "REMOVEREVISIONINFO" indicate the post
                    // failed...
                {
                    String strVerbol =
ChatVerbol.REVISED_VERB;
                    try
                    {
                        URLPost sourcePost =
(URLPost)evt.getSource();
                        if (sourcePost != null)
                        {
                            String function =
sourcePost.getFunction();
                            if (function != null &&
function.equals(URLPost.REMOVEREVISIONINFO))
                                strVerbol =
ChatVerbol.COMPLETEDREVISION_VERB;
                        }
                    }
                    catch (ClassCastException e)
                    {
                        e.printStackTrace();
                    }
                    notifyModelsOfNewLocation(strVerbol,
m_ibookedit.currentUrl);        // this could be the
URL in the POST ?
                    break;
                }
            }
        }

/*******************************************************
     * handling a LoginEvent (LoginComplete)
     */
    private void handleLoginEvent(LoginEvent evt)
    {
        /*
        m_strIbookPassword = "";
        // this is *REALLY* ugly. We use the source of
the event to find the entry field contents
        // on the dialog boxes. Bad,bad ,bad, I know,
but ibook doesn't save the current password anywhere.
        try
        {
            ibookreg2 reg = (ibookreg2)evt.getSource();
            IDD_DIAGRES dlg = reg.dlg;
            m_strIbookPassword =
dlg.IDC_PASSWORD.getText();
        }
        catch (Exception e)
        {
            e.printStackTrace();    // problem
obtaining password field - ibook controller may have
changed
            // In V1.1 using the HttpServer - this
doesn't work so ibook chat rooms are broken here too.
```

```
            }
            */
            logoutIbookChat();                  // log out of
any existing chat room - if we've logged on again, it
could be
                                                // to change
user names within the same ibook
            handleNavigateComplete();           // handle
change of ibook site
        }

/*******************************************************
     * handle navigation to a new page. Need to tell
the current ibook chat room
     * what is going on.
     */
    private void handleNavigateComplete()
    {
        // need to determine the current ibook and
current page, and update the
        // chat model/ui if necessary.

        // end up getting the current URL and ibook
from the ibookedit directly,
        // rather from this event, because too hard to
unpick the url.
        String strIbook = m_ibookedit.currentIbook;
        String strUrl = m_ibookedit.currentUrl;
        String strHost = "";
        try
        {
            URL url = new URL(strUrl);
            strHost = url.getHost();
        }
        catch (MalformedURLException e)
        {
        }

        log("*** Chat.navigatecomplete() "
                + " host=" + strHost + ", ibook=" +
strIbook + ", user=" + m_ibookedit.contributor
                + ", isibook=" +
m_ibookedit.isIBook(strHost));
        // if either the ibook or the server has
changed, we need to change our model
        if (!m_ibookedit.isIBook(strHost))
            strHost = "";

        if (!strHost.equals(m_strIbookCurrentHost) ||
!strIbook.equals(m_strIbookCurrentIbook))
        {
            // if we have a legitimate ibook host, then
look for a new connection
            // otherwise, we stick with the "old"
connection. This means you can
            // browse away from an ibook page, and
remain in the same chat room.
            if (!strHost.equals("")) // ibook could be
empty for Root ibook
            {
                switchIbookChatRoom(strHost, strIbook);

            }
        }

    }
```

```
/*****************************************************
     * Request to change to a new ibook chat room -
given the host and ibook.

*****************************************************/
    private void switchIbookChatRoom(String strHost,
String strIbook)
    {
        log("ibookchat - request switching to host=" +
strHost + " ibook=" + strIbook);

        ServerConnectionInfo info =
getChatServerForIbook(strHost, strIbook,

getCurrentIbookUser(),

getIbookPassword());
        if (info == null)           // no ibook config
page - so NO ibook host
        {
            return;     // nothing to do
        }

        // automatically fill in the nickname, etc.
        info.m_strNick = getCurrentIbookUser();
        info.m_strUser = info.m_strUser;
        info.m_strPass = getIbookPassword();

        // important to join the new channel, before
closing the existing one, so if
        // the channels are on the same server, we get
to re-use the connection - otherwise it
```

```
        // would QUIT the connection to the server and
re-open it un-neccessarily.

        ServerConnection conn =
ServerConnection.getServerConnection(info);
        ChatModel newModel =
ChatModel.joinChannel(conn, getChannelForIbook(strHost,
strIbook));

        // set the host and ibook channel we're
currently in
        m_strIbookCurrentHost = strHost;
        m_strIbookCurrentIbook = strIbook;

        setChatIbookModel(newModel);   // update the
chat model

    }

/*******************************************************
     * Changes the current chat ibook model.
Closes/removes the previous ibook
     * model.
     */
    private void setChatIbookModel(ChatModel newModel)
    {
        if (m_ibookchatmodel != null)
        {
            m_ibookchatmodel.close();      // close
down the connection
            m_ibookchatmodel = null;
        }
```

```
            m_ibookchatmodel = newModel;

            // if there's a UI around, set this to look at
    the new model - this might be a null model
            if (m_ibookchatui != null)
            {
                m_ibookchatui.setModel(newModel);  //
    should update our title and location
            }

            // if we've got a model, ask it to connect to
    the server
            if (m_ibookchatmodel != null)
            {

    m_ibookchatmodel.getChannel().getServerConnection().con
    nect();  // ask to connect to this channel

                // and tell the model where we currently
    are
                String strUrl = getIbookUrl();
                if (strUrl.length() > 0)
                {

    m_ibookchatmodel.sendVerbolNavigationMessage(ChatVerbol
    .BROWSES_VERB, strUrl);
                }
            }
            // lastly if, we've changed model, we may also
    need to update the controller title.
            updateControllerTitle();

        }
```

```
/*****************************************************
     * Leaves the current ibook chat room and re-sets
all variables.
     */
    private void logoutIbookChat()
    {
        setChatIbookModel(null);
        m_strIbookCurrentHost = null;
        m_strIbookCurrentIbook = null;
    }

/*****************************************************
     * Implements a chat property page - just done as
an example for now
     * for testing
     */
    public class ChatPropertyPage extends
com.fs.util.PropertyPage
    {
        public ChatPropertyPage()
        {
            super();
            // just create a label to demo we can
extend property pages
            JLabel label = new JLabel("Chat properties
can go here");
            add("Center", label);
        }

        public boolean onApply()
        {
            return true;
            }
        }

        private void handleIBookEditEvent(IbookEditEvent
    evt)
        {
            int nType = evt.getID();
            switch (nType)
            {
                case IbookEditEvent.CONFIGURE:
                    com.fs.util.PropertySheet sheet =
    (com.fs.util.PropertySheet)evt.getObject();
                    ChatPropertyPage page = new
    ChatPropertyPage();
                    sheet.addPage("Chat", null, page);
                    break;

                case IbookEditEvent.MENU_ACTION:
                    String strAction = evt.getURL();
    // get the text of the event
                    if (strAction.equals("CHAT_MEETING"))
                    {
                        onMenuChatShow();
                    }
                    else if
    (strAction.equals("CHAT_IBOOK"))
                    {
                        onMenuChatIbook();
                    }
                    else if (strAction.equals("LOGOUT"))
                    {
                        logoutIbookChat();
                    }
```

```
                        break;

                    case IbookEditEvent.LOAD_PROPERTIES:
                    {
                        Properties props =
(Properties)evt.getObject();

                        m_strChatServer = getProperty(props,
"chatext.server", ""); //default to nowhere
                        m_strChatNick = getProperty(props,
"chatext.nick", "");
                        m_strChatChannel = getProperty(props,
"chatext.channel", "");
                        m_nChatPort = getProperty(props,
"chatext.port", 6667);
                        m_strChatPassword = getProperty(props,
"chatext.password", "");
                        m_strChatUser = getProperty(props,
"chatext.user", "");
                        m_strChatRealName = getProperty(props,
"chatext.realname", "");
                        m_strChatDefServer = getProperty(props,
"chatext.defserver", "");  // default server to use for
ibook stuff
                        m_bShowVerbol = getProperty(props,
"chatext.showverbol", false);
                        setTrustedUsers(getPropertyEnum(props,
"chatext.trusted", ';'));

                        // position of ibook window
                        m_rectIbook.x = getProperty(props,
"chatext.ibook.rect.x", 0);
```

```
                    m_rectIbook.y = getProperty(props,
"chatext.ibook.rect.y", 0);
                    m_rectIbook.width = getProperty(props,
"chatext.ibook.rect.wd", 0);
                    m_rectIbook.height = getProperty(props,
"chatext.ibook.rect.ht", 0);
                    break;
                }

                case IbookEditEvent.SAVE_PROPERTIES:
                {
                    Properties props =
(Properties)evt.getObject();

                    setProperty(props, "chatext.server",
m_strChatServer);
                    setProperty(props, "chatext.nick",
m_strChatNick);
                    setProperty(props, "chatext.user",
m_strChatUser);
                    setProperty(props, "chatext.realname",
m_strChatRealName);
                    setProperty(props, "chatext.channel",
m_strChatChannel);
                    setProperty(props, "chatext.port",
m_nChatPort);
                    setProperty(props, "chatext.password",
m_strChatPassword);
                    setProperty(props, "chatext.defserver",
m_strChatDefServer);
                    setProperty(props,
"chatext.showverbol", m_bShowVerbol);
```

```
                setPropertyEnum(props,
"chatext.trusted", ';', getTrustedUsers());

                // if we got an ibook chat room, then
save it's position
                if (m_ibookchatui != null)
                {
                    m_rectIbook =
m_ibookchatui.getFrame().getBounds();

                    setProperty(props,
"chatext.ibook.rect.x", m_rectIbook.x);
                    setProperty(props,
"chatext.ibook.rect.y", m_rectIbook.y);
                    setProperty(props,
"chatext.ibook.rect.wd", m_rectIbook.height);
                    setProperty(props,
"chatext.ibook.rect.ht", m_rectIbook.width);
                }

                break;
            }
        }
    }

/*******************************************************
     * menu action for making the ibook chat window
visible. Not implemented
     * at present.
     */
    private void onMenuChatIbook()
    {
        // have we created the GUI yet ?
```

```
            if (m_ibookchatui == null)
            {
                ChatModel model = m_ibookchatmodel;
                m_ibookchatui = new ChatInstance(this,
model, ChatInstance.MODE_IBOOK);      // create the new
instance
                // position it where it was when we closed
last time
                if (!m_rectIbook.isEmpty())
                {

m_ibookchatui.getFrame().setBounds(m_rectIbook);
                }
                updateControllerTitle();
            }
            m_ibookchatui.getFrame().setVisible(true);
// make it visible
        }


/*****************************************************
         * User has asked to view the chat show window
         * Must prompt them for something to log into...
         */
        private void onMenuChatShow()
        {
            ConnectionDialog dlg = new ConnectionDialog();
            ServerConnectionInfo info = new
ServerConnectionInfo();
            info.m_strServer = m_strChatServer;

            // if there's no remembered username - see if
we're currently logged on at an ibook
```

```
            String strNick = m_strChatNick;
            if (strNick.length() == 0)
                strNick = getCurrentIbookUser();
            info.m_strNick = strNick;
            info.m_nPort = m_nChatPort;

            info.m_strUser = m_strChatUser;
            info.m_strRealName = m_strChatRealName;

            dlg.setConnectionInfo(info);
            dlg.setChannel(m_strChatChannel);
            // NB - we created the ConnectionDialog as non-
modal, so we effectively return immediately.
            dlg.setVisible(true);
        }

/*******************************************************
         * Sets the configuration chat channel (so it's
remembered for next time)
         */
        public void setConfigChatChannel(String strChannel)
        {
            m_strChatChannel = strChannel;
        }

/*******************************************************
         * Sets the stored configuration parameters to be
those in the specified connection
         * (so it's remembered for next time)
         */
        public void setConfigChatInfo(ServerConnectionInfo
info)
        {
            m_strChatServer = info.m_strServer;
            m_strChatNick = info.m_strUser;
            m_strChatNick = info.m_strNick;
            m_nChatPort = info.m_nPort ;
            m_strChatUser = info.m_strUser;
            m_strChatRealName = info.m_strRealName;
            m_strChatPassword = info.m_strPass;
        }

    /*******************************************************
         * Tells the ibook controller to execute the
    specified Verbol

    *******************************************************/
        public void ibook_executeVerbol(String strText)
        {
            if (m_ibookedit == null)
                return;          // no ibook - nothing to do
            verbolServerEvent evt = new
    verbolServerEvent(this, verbolServerEvent.VERBOL);
            evt.setContributor("chat");
            evt.setVerbol(strText);
            m_ibookedit.verbolServerStatement(evt);
        }

    /*******************************************************
         * Tell the ibook controller to navigate to the
    specified page, forcing a refresh
         * if necessary.

    *******************************************************/
        public void ibook_navigateTo(String url, boolean
    bForceRefresh)
```

```
        {
            log("ibook chat - navigateTo " + url + "
refresh=" + bForceRefresh);
            if (m_ibookedit == null)
                return;

            url = url.trim();
            if (!url.equals(getIbookUrl()))
            {
                log("asking ibook to goNavigate(" + url +
")");
                m_ibookedit.browser.goNavigate(url, " ");
            }
            else if (bForceRefresh)
            {
                log("asking ibook to refresh & goNavigate("
+ url + ")");
                //m_ibookedit.browser.goNavigate(url, " ");
                m_ibookedit.browser.refresh();  // refresh
the current page
            }
            else
                log("ignoring request to navigate To " +
url);
        }


/*****************************************************
     * launch thread to preload classes we need for the
UI
     */
    static void preloadClasses()
    {
            Thread t = new BackgroundLoader();
            t.start();
        }

        private void setProperty(Properties props, String
    name, String value)
        {
            if (value != null)
            {
                props.put(name,value);
            }
            else
            {
                props.remove(name);
            }
        }

        private void setProperty(Properties props, String
    name, int nValue)
        {
            props.put(name, Integer.toString(nValue));
        }

        private void setProperty(Properties props, String
    name, boolean bValue)
        {
            props.put(name, bValue ?
    Boolean.TRUE.toString() : Boolean.FALSE.toString());
        }

        private String getProperty(Properties props, String
    strName, String strDefault)
        {
            String str = (String)props.get(strName);
            if (str == null)
                return strDefault;
            else
                return str;
        }

        private boolean getProperty(Properties props,
    String strName, boolean bDefault)
        {
            String str = (String)props.get(strName);
            if (str == null)
                return bDefault;
            else
                return Boolean.valueOf(str).booleanValue();
        }


        private int getProperty(Properties props, String
    strName, int nDefault)
        {
            String str = (String)props.get(strName);
            if (str == null)
                return nDefault;
            else
            {
                try
                {
                    return Integer.parseInt(str);
                }
                catch (NumberFormatException e)
                {
                    return nDefault;
            }
        }
    }

/*****************************************************
     * Obtains an enumeration of string properties.
Returns an enumeration

*****************************************************/
    private Enumeration getPropertyEnum(Properties
props, String strName, char chDelimiter)
    {
        Vector v = new Vector();
        String str = (String)props.get(strName);
        if (str != null)
        {
            StringTokenizer tok = new
StringTokenizer(str, String.valueOf(chDelimiter));
            while (tok.hasMoreTokens())
                v.addElement(tok.nextToken());
        }
        return v.elements();
    }


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

     * Sets an enumeration of string properties

*****************************************************/
    private void setPropertyEnum(Properties props,
String strName, char chDelimiter, Enumeration enum)
    {
            StringBuffer buf = new StringBuffer();
            while (enum.hasMoreElements())
            {
                String str = (String)enum.nextElement();
                if (buf.length() > 0)
                    buf.append(chDelimiter);
                buf.append(str);
            }
            setProperty(props, strName, buf.toString());
        }

    /*****************************************************
    * Adds a specified user into the list of trusted users

    *****************************************************/
        public void addTrustedVerbolUser(String strUser)
        {
            m_hashTrustedVerbolUsers.put(strUser, strUser);
    // add this user into the hashtable
        }

    /*****************************************************
         * Returns true if we trust this user to execute
    Verbol on us without asking

    *****************************************************/
        public boolean isUserTrusted(String strUser)
        {
            return
    m_hashTrustedVerbolUsers.containsKey(strUser);
        }
```

```
/*******************************************************
      * Get a enumeration of the trusted users

*******************************************************/
    public Enumeration getTrustedUsers()
    {
        return m_hashTrustedVerbolUsers.keys();
    }

/*******************************************************
* Sets the set of currently accepted users

*******************************************************/
    public void setTrustedUsers(Enumeration enum)
    {
        m_hashTrustedVerbolUsers.clear();
        while (enum.hasMoreElements())
        {

addTrustedVerbolUser((String)enum.nextElement());
        }
    }

/*****************************************************
      * Sends the specified text message to all the
channels (instances) we have
      * created. This is used to ensure all chat rooms
get the same Verbol. NB -
      * we could do this more efficiently, both for the
server and the client, by
      * just sending a single message. In future may
wish to re-think how multiple
```

```
        * chat rooms should be presented.

    ******************************************************/
        public void sendMessageToAllChannels(String
    message)
        {
            Enumeration enum = getModels();
            while (enum.hasMoreElements())
            {
                ChatModel model =
    (ChatModel)enum.nextElement();
                model.sendMessageToChannel(message);
            }
        }

        private void notifyModelsOfNewLocation(String verb,
    String url)
        {
            Enumeration enum = getModels();
             while (enum.hasMoreElements())
             {
                 ChatModel model =
    (ChatModel)enum.nextElement();
                 model.sendVerbolNavigationMessage(verb,
    url);
             }
         }

    /******************************************************
        * Returns an enumeration of all the models we have
        * TODO - clean this up.
        */
        private Enumeration getModels()
        {
            Vector v = new Vector();
            for (int i = 0; i < m_vecInstances.size(); i++)
            {
                ChatInstance inst =
    (ChatInstance)m_vecInstances.elementAt(i);
                ChatModel model = inst.m_model;
                if (model != m_ibookchatmodel)           //
    skip the ibook chat room
                    v.addElement(model);
            }
            if (m_ibookchatmodel != null)
                v.addElement(m_ibookchatmodel);
            return v.elements();
        }


    /******************************************************
         * Returns the user currently logged onto the ibook
         */
        public String getCurrentIbookUser()
        {
            String str = null;
            if (m_ibookedit != null)
            {
                str = m_ibookedit.contributor;
                // bodge - if not logged in yet, then ibook
    has the username "default"
                if (str.equalsIgnoreCase("default"))
                    str = null;
            }
            if (str == null)
                str = "";
            return str;
        }

    /*******************************************************
    * Encapsulate logging from the chat client - maybe have
    a useful log later.
         */
        public void log(String str)
        {
            System.out.println(str);
        }

    /*******************************************************
    * This method attempts to use reasonable logic to
    update the nickname displayed by
         * the ibook controller.
         */
        public void updateControllerTitle()
        {
            // What is the correct title to use ?
            Hashtable hash = new Hashtable();
            Enumeration enum = getModels();

            // iterate thru the nicks in use and put them
    in a hash table so we end up with a unique list
            while (enum.hasMoreElements())
            {
                ChatModel model =
    (ChatModel)enum.nextElement();
                // if it's the ibook chat model, then we
    ONLY want to include it if the chat gui is visible.
                if (model.isNull())
```

```
                    continue;           // ignore the null
model !

                if (model == m_ibookchatmodel
                    && (m_ibookchatui == null ||
!m_ibookchatui.getFrame().isVisible()))
                    continue;

                String nick = model.getMyNick();
                if (nick != null)
                    hash.put(nick, nick);
            }

            // have a hash table of nickname - we display
them all (unique ones).
            StringBuffer buf = new StringBuffer();
            Enumeration enumNicks = hash.keys();
            while (enumNicks.hasMoreElements())
            {
                String nick = (String)
enumNicks.nextElement();
                if (buf.length() > 0)
                    buf.append(",");
                buf.append(nick);
            }
            log("Changing window title to " +
buf.toString());
            if (m_ibookedit != null)

m_ibookedit.setChatNickname(buf.toString());
        }
    }
```

```
//Source file:
./chat/ChatExtension/src/ChatInstance.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++
// has to be in root package to access ibook stuff.

import com.fs.chatclient.*;
import com.fs.chatextension.*;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.table.*; // for table model, etc
import javax.swing.border.Border;
import javax.swing.plaf.basic.BasicArrowButton;
import javax.swing.event.ListDataEvent;
import javax.swing.event.ListDataListener;
import javax.swing.SwingUtilities;
import com.fs.util.PropertySheet;               // for
options...
import com.fs.util.BrowserLauncher;

import java.net.URL;                            // for
loading resources
import javax.swing.text.Document;
import java.io.*;

/**
 * Represents an instance of a chat room as seen from
the perspective of the ibook
 * controller. Consists of an embedded GUI, a specified
chat connection, etc.
 * $Date: 2002/04/25 17:06:58 $
 * $Author: markc $
 * $Revision: 1.43 $
 *
 */

public class ChatInstance implements
ChatModelVerbolListener, ActionListener,
ListDataListener
{
    static final boolean USE_TEXTPANE = true;

    ChatExtension       m_ext;               //
extension (there's only ever 1 anyway)
    ChatModel           m_model;             // model
containing chat data

    private ChatPopupWindow m_popupWindow = null;

    private JFrame      m_frame;             // frame
for this instance

    private JSplitPane  m_paneSplit;         // splitter
pane
    private JPanel      m_paneMain;          // main
chat panel
    private JTextField  m_fieldInput;        // input
text area
    private JTable      m_tableUsers;        // user
list
    private JTextArea   m_labelNoChannel;    // label to
display iff there is no channel
```

```
    // use either the list OR the text pane - not both
!
    private ChatTextPane m_textChat;          // text
area for chat messages
    private JList       m_listChat;           // list
area for chat messages

    private JScrollPane m_textScrollPane;    // scroll
pane for the text control
    private JScrollPane m_tableScrollPane;   // scroll
pane for the Users table
    private JButton     m_btnSend;            // send
button
    private JButton     m_btnMenu;            // menu
button
    private JButton     m_btnHelp;            // help
button
    private JButton     m_btnSendDropdown;   // drop
down send button
    private boolean     m_bSplitHorizontal = true;
    private boolean     m_bShowFriendly = false;     //
when set to true, freindly messages are displayed.

    ChatIconFrame       m_chatIconFrame = null; // the
chat icon frame for this window (null if none)
    // menu-items
    private JMenuItem m_miFollowPerson;
    private JMenuItem m_miFollowStop;
    private JMenuItem m_miGoThere;
    private JMenuItem m_miSendPrivate;
    private JMenuItem m_miSendVerbol;
    private JMenuItem m_miRefreshUsers;
    private JMenuItem m_miFlipLayout;
```

```
        private JMenuItem m_miHistory;
        private JMenuItem m_miHideWindow;
        private JMenuItem m_miOptions;
        private JMenuItem m_miChatIcons;
        private JMenuItem m_miClearWindow;
        private JMenuItem m_miPrint;

        // Modes are distinguised as:
        // MEETING
        //  * user can connect to any chat server
        //  * server connection is owned by the window,
    when the window closes, so does the server connection
        //  * window is initially visible.
        //  * can have multiple chat meetings

        // IBOOK
        // * user connects to the server for the current
    ibook and logs onto that chat room
        // * window not intially visible. Server connection
    survives without window visible.
        // * only 1 single IBOOK context
        // * connection is maintained if user closes window
        // (ibook mode not implemented yet)


        static final int MODE_MEETING = 1;       // meeting
    mode - connection is "owned"instance
        static final int MODE_IBOOK = 2;         // ibook
    mode - connection is
        private int m_nMode = MODE_MEETING;
    static public ChatInstance
newInstance(ChatExtension ext, ServerConnectionInfo
info, String strChannel)
    {
        // connect to the server
        ServerConnection server =
ServerConnection.getServerConnection(info);
        ChatModel model = ChatModel.joinChannel(server,
strChannel);
        server.connect();

        ChatInstance inst = new ChatInstance(ext,
model, MODE_MEETING);
        inst.m_frame.setVisible(true);
        return inst;
    }

/*******************************************************
* ChatInstance constructor. The chat UI must have a
model to refer to.
     *
     */
    public ChatInstance(ChatExtension ext, ChatModel
model, int nMode)
    {
        if (model == null)
            model = ChatModel.getNullChannel();

        m_model = model;
        m_ext = ext;
        m_nMode = nMode;
        model.setVerbolListener(this);
        constructControls();
        // find out if the ibook has a location and use
it to tell everyone our current location
        if (ext != null)
        {
            ext.addInstance(this);      // add this to
the collection the ChatExtension knows about !

            String strUrl = ext.getIbookUrl();
            if (strUrl.length() > 0)
            {

m_model.sendVerbolNavigationMessage(ChatVerbol.BROWSES_
VERB, strUrl);
            }
        }
        updateFrameTitle();     // so the title is
correct
    }

/*****************************************************
    * Sets the ChatModel this UI instance should be
displaying details for.
    * Needs to handle updating the title, etc.

*****************************************************/
    public void setModel(ChatModel model)
    {
        // if we're not given a - use the null wrapper
one
        if (model == null)
            model = ChatModel.getNullChannel();
        if (model == m_model)
            return;          // same - not changed -
nothing to do !

        m_model = model;
        model.setVerbolListener(this);  // set the call
back for the UI events

        updateFrameTitle();


m_tableUsers.setModel(m_model.getUsersTableModel());
        if (!USE_TEXTPANE)

m_listChat.setModel(m_model.getMessageListModel());
    }

/*******************************************************
     * Returns the frame for this UI - used by callers
to make visible, etc
     */
    public JFrame getFrame()
    {
        return m_frame;
    }

/*******************************************************
     * Construct the splitter pane and other controls
     */
    private void constructControls()
    {
        m_paneMain = new JPanel();
        m_paneMain.setLayout(new BorderLayout());
        // create the split pane and put this in the
border layout with an entry field.
        createSplitPane();

        m_fieldInput = new VTextField(40);
        m_fieldInput.setText("");
//        m_fieldInput.setToolTipText("Enter your
message here");   // tried this and it was annoying !
        m_fieldInput.addActionListener(this);

        JPanel pane = new JPanel();
        pane.setLayout(new BoxLayout(pane,
BoxLayout.X_AXIS));
        JLabel label = new JLabel("Message:");
        pane.add(label);
        pane.add(m_fieldInput);

        // get the height of the label to use for
buttons.
        Dimension sizeLabel = label.getPreferredSize();

        m_btnSend = createButtonHelper("Send",
sizeLabel.height, pane, "images/btnsend.gif");

        m_btnSend.addMouseListener(new MouseAdapter()
            {
                public void mousePressed(MouseEvent e)
                {
                    if (e.isPopupTrigger())
                    {
```

```
                                JPopupMenu menu =
buildSendPrivateMenu();

menu.show(m_btnSend.getParent(), m_btnSend.getX(),
m_btnSend.getY());
                        }
                    }
                });

        // add the drop down message for buttons
        m_btnSendDropdown = createButtonHelper("Send
Private", sizeLabel.height, pane,
"images/btndrop.gif");

        // other buttons (help, etc)
        m_btnMenu = createButtonHelper("Menu",
sizeLabel.height, pane, null);
        m_btnHelp = createButtonHelper("Help",
sizeLabel.height, pane, "images/btnhelp.gif");

        m_paneMain.add(pane, BorderLayout.SOUTH);

        // build them into a frame...
        m_frame = new JFrame("Chat Frame");
        m_frame.getContentPane().setLayout(new
BorderLayout());
        m_frame.getContentPane().add(m_paneMain,
BorderLayout.CENTER);

        m_frame.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
{onWindowClose();}
```

```
            public void windowIconified(WindowEvent e)
{onWindowIconified(true); }

            public void windowDeiconified(WindowEvent
e) {onWindowIconified(false); }
        });

        m_frame.pack();

        // create the popup window for this window
(only gets shown if this window is hidden)
        m_popupWindow = new ChatPopupWindow();
        m_popupWindow.setFrame(m_frame);

        m_labelNoChannel = new JTextArea("No ibook chat
room is available. You may not be at an ibook site, or
are not logged in");
        m_labelNoChannel.setEnabled(false);
        m_labelNoChannel.setLineWrap(true);
        m_labelNoChannel.setWrapStyleWord(true);
        m_labelNoChannel.setAlignmentX(0.0f);
        m_labelNoChannel.setAlignmentY(0.0f);
        m_labelNoChannel.setEditable(false);
        m_labelNoChannel.setFont(m_frame.getFont());
// use the frame's font (a bit bigger)
        m_labelNoChannel.setPreferredSize(new
Dimension(300, 200));

    }

    // Creates a button on the bottom row.
    private JButton createButtonHelper(String
strButtonLabel, int nHeight, JPanel parent, String
strIcon)
    {
        JButton btn = null;
        if (strIcon != null)
        {
            ImageIcon icon = loadImageIcon(strIcon);
            if (icon != null)
            {
                btn = new JButton(icon);
                btn.setToolTipText(strButtonLabel);  //
use the text as a tool tip
            }
        }
        if (btn == null)        // ie not already done
an icon
        {
            btn = new JButton(strButtonLabel);
        }
        // make the margin on the button smaller -
seems to default to 14 pixels Left & Right.
        // The buttons are Huge !
        btn.setMargin(new Insets(2,2,2,2));
        Dimension sizeBtn = btn.getPreferredSize();
        sizeBtn.height = nHeight;
        btn.setPreferredSize(sizeBtn);
        btn.addActionListener(this);
        parent.add(btn);
        return btn;
    }
```

```
        // attempt to hook into iconfied status - but still
    means there is no JDK 1.1 programmatic way to
        // restore a window
        private void onWindowIconified(boolean bIconified)
        {
            //m_bIconified = bIconified;
        }


        private void onWindowClose()
        {
            if (m_nMode == MODE_MEETING)
            {
                m_frame.dispose();          // get rid of
    the window

                // leave the channel - send a PART message
                m_model.close();
                m_ext.removeInstance(this);
    // remove from list of instances
                if (m_ext == null ||
    m_ext.isRunningStandalone())    // exit
                    System.exit(0);
            }
            else if (m_nMode == MODE_IBOOK)
            {
                // user has elected to close the window -
    so we don't want the yellow
                // box popping up each time. They will only
    get this now if the elected
                // to hide the window explicitly.
                m_bShowFriendly = false; // don't show the
    friendly messages
```

```
                m_frame.setVisible(false);          // just
    hide the frame - all other behaviour continues
            }
            m_ext.updateControllerTitle();
        }

    /*****************************************************
         * Handles the enter key being pushed on the text
    box, and other key actions

    *****************************************************/
        public void actionPerformed(ActionEvent evt)
        {
            try
            {
                Object source = evt.getSource();
                String action = evt.getActionCommand();
                if (source == m_fieldInput || source ==
    m_btnSend)
                {
                    onSendMessage();
                }
                else if (source == m_btnMenu)
                {
                    JPopupMenu menu = buildPopupMenu();
                    menu.show(m_btnMenu.getParent(),
    m_btnMenu.getX(), m_btnMenu.getY());
                }
                else if (source == m_btnHelp)
                {
                    launchHelp("ct_main_w.htm");
                }
                else if (source == m_btnSendDropdown)
                {
                    JPopupMenu menu =
buildSendPrivateMenu();

menu.show(m_btnSendDropdown.getParent(),
m_btnSendDropdown.getX(), m_btnSendDropdown.getY());
                }
                else if (source == m_miSendPrivate)
                {
                    onSendPrivateToSelected();
                }
                else if (source == m_miSendVerbol)
                {
                    onSendVerbolToSelected();
                }
                else if (source == m_miFollowPerson)
                {
                    onFollowSelectedNick();
                }
                else if (source == m_miFollowStop)
                {
                    onFollowStop();
                }
                else if (source == m_miGoThere)
                {
                    onGoThere();
                }
                else if (source == m_miRefreshUsers)
                {
                    onRefreshUsers();
                }
                else if (source == m_miFlipLayout)
                {
                    m_bSplitHorizontal =
!m_bSplitHorizontal;
                    replaceSplitPane(m_bSplitHorizontal);
                    m_frame.validate();
                }
                else if (source == m_miHistory)
                {
                    onHistory();
                }
                else if (source == m_miHideWindow)
                {
                    onHideWindow();
                }
                else if (source == m_miOptions)
                {
                    onOptions();
                }
                else if (source == m_miChatIcons)
                {
                    onChatIcons();
                }
                else if (source == m_miClearWindow)
                {
                    onClearWindow();
                }
                else if (source == m_miPrint)
                {
                    onPrint();
                }
                else if (action.startsWith("SEND "))
                {

parseMenuSendCommand(action.substring(5));
```

```
                }
            }
            catch (Throwable e)
            {
                reportException(e);

            }
        }

        public static void reportException(Throwable e)
        {
            // an unexpected exception in the UI - catch it
    and display it.
            StringWriter sw = new StringWriter();
            PrintWriter pw = new PrintWriter(sw);
            pw.print("An unexpected error has occurred,
    please contact Family Systems technical support and
    include the following details\n\n");
            e.printStackTrace(pw);
            JOptionPane.showMessageDialog(null,
                                        sw.toString(),
                                        "Unexpected Error",

    JOptionPane.ERROR_MESSAGE);
        }

    /*******************************************************
          * This only handles internally constructed
    commands - NOT free form text.
          * Format is:
          * SEND <to> <message>
          * <to> can be * if it should be the currently
    selected Nick.
```

```
        * if <message> is not present, we send the
contents of the text box

***************************************************************/
    private void parseMenuSendCommand(String strAction)
    {
        // next is either a Nick to send to, or "*" to
send to the selected.
        StringTokenizer2 tok = new
StringTokenizer2(strAction, " ");
        String to = tok.nextToken();
        if (to.equals("*"))
            to = getSelectedNick();

        if (!tok.hasMoreTokens())
            onSendPrivateToNick(to);
        else
            m_model.sendOutgoingMessage(to,
tok.getRemainder());
    }


    private void onSendMessage()
    {
        String strText = m_fieldInput.getText();
        if (strText.length() == 0)
        {
            JOptionPane.showMessageDialog(m_frame,
"Please enter the text of your message in the text
box");
            return;
        }
        // special case for dumping debug info to help
diagnose problems
        if (strText.equals("/debug"))
            dumpDebug();
        else
            m_model.sendMessageToChannel(strText);
        m_fieldInput.setText("");        // clear the
text
    }

    //
*******************************************************
    // START Implementation of ChatModelVerbolListener
interface
    //
*******************************************************

/******************************************************
     * sends a verbol execution string to the ibook
that launched this extension
     * if there was an extension
     */
    private void executeVerbol(String str)
    {
        str = str.trim();
        if (str.length() == 0)
            return;         // protection against Empty
string - shouldn't happen !

        System.out.println("execute verbol: " + str);
        if (m_ext != null)      // no extension -
nothing to do.
            m_ext.ibook_executeVerbol(str);
    }

    public void navigateTo(String url, boolean
bRefresh)
    {
        url = url.trim();
        if (url.length() == 0)
            return;         // protection against Empty
string - shouldn't happen !
        if (m_ext != null)
            m_ext.ibook_navigateTo(url, bRefresh);
    }


    public void textMessage(String strFrom, String
strMessage, int nMsgType)
    {
        // we can get an incoming textMessage, while
we're in the process of creating the
        // controls for this frame. If this happens, we
ignore it. We could also be
        // in the position of knowing we're being made
visible, so don't want to display the
        // message either.

        // can't test for iconify in JDK 1.1....
        System.out.println("textMessage() nMsgType=" +
nMsgType);
        if (USE_TEXTPANE && m_textChat != null)
        {
            m_textChat.addMessage(strFrom, strMessage);
            scrollToEnd();
        }
```

```
            if (m_bShowFriendly && (m_frame != null &&
    !m_frame.isVisible()) /*|| m_frame.getState() ==
    Frame.ICONIFIED*/)
            {
                if (nMsgType == ChatModel.MSGTYPE_INCOMING
                    || nMsgType ==
    ChatModel.MSGTYPE_SERVERNICE)
                {
                    showFriendlyMessage(strFrom,
    strMessage);
                }
            }
        }

        //
    *******************************************************
    // END Implementation of ChatModelVerbolListener
    interface
        //
    *******************************************************

    /******************************************************
         * Display a concise version of the message in the
    text window. It can't be
         * very long, and want to interpret Verbol in a
    friendly manner.
         */
        private void showFriendlyMessage(String strFrom,
    String strMessage)
        {
          StringTokenizer2 tok = new
StringTokenizer2(strMessage, " ");
          if (!tok.hasMoreTokens())
               return;          // no tokens - give up

          if
(!tok.nextToken().equals(ChatVerbol.VERBOL_INDICATOR))
          {
               if (strMessage.length() > 80)
                    strMessage = strMessage.substring(0,
77) + "...";
          }
          else
          {
               if (!tok.hasMoreTokens())
                    return;
               String strVerb = tok.nextToken();

               if
(strVerb.equals(ChatVerbol.REVISING_VERB))
               {
                    strFrom += " is revising";
                    strMessage = tok.getRemainder();
               }
               else if
(strVerb.equals(ChatVerbol.REVISED_VERB))
               {
                    strFrom += " has revised";
                    strMessage = tok.getRemainder();
               }
               else if
(strVerb.equals(ChatVerbol.BROWSES_VERB))
               {
                    return;         // ignore this
                }
                else if
(strVerb.equals(ChatVerbol.COMPLETEDREVISION_VERB))
                {
                    strFrom += " finished revising";
                    strMessage = tok.getRemainder();
                }
            }
            System.out.println("show friendly message" +
strFrom + " " + strMessage);
            m_popupWindow.showMessage(strFrom, strMessage);
        }

        private void createSplitPane()
        {
            if (USE_TEXTPANE)
            {
                m_textChat = new ChatTextPane();
                m_textScrollPane = new
JScrollPane(m_textChat,
JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,

JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
            }
            else
            {
                m_listChat = new
JList(m_model.getMessageListModel());

m_listChat.setSelectionMode(ListSelectionModel.SINGLE_S
ELECTION);
                m_listChat.setAutoscrolls(true);
            m_textScrollPane = new
JScrollPane(m_listChat,
JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,

JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        }
        // do we need to register with the list ?

m_model.getMessageListModel().addListDataListener(this)
;    // so we know when something has been added to the
list

        m_textScrollPane.setAutoscrolls(true);

        m_tableUsers = new
JTable(m_model.getUsersTableModel());
        m_tableUsers.setShowHorizontalLines(false);
        m_tableUsers.setShowVerticalLines(false);
        m_tableScrollPane = new
JScrollPane(m_tableUsers);

        TableColumnModel colModel =
m_tableUsers.getColumnModel();

        // specialised renderer for Nick names to add
icons
        colModel.getColumn(0).setCellRenderer(new
NickRenderer(false));

        // set widths to make it clearer
        colModel.getColumn(0).setPreferredWidth(25);
        colModel.getColumn(1).setPreferredWidth(25);
        colModel.getColumn(2).setPreferredWidth(400);
        //Add listener to components that can bring up
popup menus.
        MouseListener popupListener = new
PopupListener();
        m_tableUsers.addMouseListener(popupListener);
        if (USE_TEXTPANE)
            m_textChat.addMouseListener(popupListener);

        //Create a split pane with the two scroll panes
in it
        replaceSplitPane(m_bSplitHorizontal);
// create the split pane and make it horizontal, etc

        //Provide minimum sizes for the two components
in the split pane
        Dimension minimumSize = new Dimension(100, 50);
        m_textScrollPane.setMinimumSize(minimumSize);
        m_tableScrollPane.setMinimumSize(minimumSize);
    }


    public void replaceSplitPane(boolean bHorizontal)
    {
        JSplitPane paneSplit;
        if (bHorizontal)
            paneSplit = new
JSplitPane(JSplitPane.VERTICAL_SPLIT ,
                                    m_tableScrollPane,
m_textScrollPane);
        else
            paneSplit = new
JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                                        m_textScrollPane,
m_tableScrollPane);

            paneSplit.setOneTouchExpandable(true);
            paneSplit.setDividerLocation(100);

            //Provide a preferred size for the split pane
            paneSplit.setPreferredSize(new Dimension(400,
200));

            // if there was an existing split pane
component, remove it
            if (m_paneSplit != null)
                m_paneMain.remove(m_paneSplit);

            m_paneSplit = paneSplit;
            m_paneMain.add(paneSplit, BorderLayout.CENTER);
        }


        private static void snapTableToColumnSizes(JTable
table)
        {
            TableModel model = table.getModel();

            TableColumn column = null;
            Component comp = null;

            for (int i = 0; i < 5; i++) {
                int nMaxWidth = 0;
                column =
table.getColumnModel().getColumn(i);
```

```
                try {
                    comp = column.getHeaderRenderer().

getTableCellRendererComponent(
                                                null,
column.getHeaderValue(),
                                                false, false, 0,
0);
                    nMaxWidth =
comp.getPreferredSize().width;
                } catch (NullPointerException e) {
                    System.err.println("Null pointer
exception!");
                    System.err.println("  getHeaderRenderer
returns null in 1.3.");
                    System.err.println("  The replacement
is getDefaultRenderer.");
                }

                // get the component to render the row
values and use that.

                for (int j = 0; j < table.getRowCount();
j++)
                {
                    comp =
table.getDefaultRenderer(model.getColumnClass(i)).

getTableCellRendererComponent(
                                                table,
table.getValueAt(j, i),
                                                false, false, 0,
i);
```

```
                    int cellWidth =
comp.getPreferredSize().width;
                    if (cellWidth > nMaxWidth)
                        nMaxWidth = cellWidth;
                }

                //XXX: Before Swing 1.1 Beta 2, use
setMinWidth instead.
                column.setPreferredWidth(nMaxWidth);
            }
        }


/*******************************************************
      * Updates the title on the frame to reflect the
name of the channel

*******************************************************/
    private void updateFrameTitle()
    {
        Channel channel = m_model.getChannel();
        String strTitle = (m_nMode == MODE_IBOOK ?
"ibook Chat Room - " : "Chat Room - ");

        if (channel == null)
        {
            strTitle += "no chat room available";
            showMainPanel(false);
        }
        else
        {
            strTitle += channel.getName().substring(1);
            showMainPanel(true);
            }
             m_frame.setTitle(strTitle);
        }


/******************************************************
     * Swaps out the content pane to be either the full
chat panel, or
     * the label saying "no channel Available"
     */
    private void showMainPanel(boolean bMainPanel)
    {
        // should we show the main panel, OR the label.
        // look for either m_paneMain, or
m_labelNoChannel in the components
        Container content = m_frame.getContentPane();
        Component comps[] = content.getComponents();
        for (int i = 0; i < comps.length; i ++)
        {
            if (comps[i] == m_paneMain)
            {
                if (!bMainPanel)
                {
                    content.remove(i);
                    content.add(m_labelNoChannel,
BorderLayout.CENTER);
                    break;
                }
                else
                    return;
            }
            else if (comps[i] == m_labelNoChannel)
            {
                    if (bMainPanel)
                    {
                        content.remove(i);
                        content.add(m_paneMain,
BorderLayout.CENTER);
                        break;
                    }
                    else
                        return;
                }
            }
            m_frame.pack();
        }

/*******************************************************
     * helper function for adding menu items
     */
    private JMenuItem addMenuHelper(JPopupMenu popup,
String strText)
    {
        JMenuItem item = new JMenuItem(strText);
        popup.add(item);
        item.addActionListener(this);
        return item;
    }

/*******************************************************
     * Builds the popup menu - based on context
     */
    private JPopupMenu buildPopupMenu()
    {
        // build the menu contents based on what is
selected.
```

```
        int nRow = m_tableUsers.getSelectedRow();

        //Create the popup menu.
        JPopupMenu popup = new VPopupMenu();

        String strMyNick = m_model.getMyNick();
        String strNick = getSelectedNick();    // get
the nickname

        // if there's a selected name, which isn't me
        if (strNick.length() > 0 &&
!strNick.equals(m_model.getMyNick()))        // if it's
not me
        {
            m_miGoThere = addMenuHelper(popup, "Browse
to the same URL as " + strNick);   // option to go to
their URL
            if
(!strNick.equals(m_model.m_strNickToFollow))      //
option to follow someone
            {
                m_miFollowPerson = addMenuHelper(popup,
"Follow " + strNick);
            }
        }

        // at brian's request, stop following comes
directly under following
        if (m_model.m_strNickToFollow.length() > 0)
        {
            m_miFollowStop = addMenuHelper(popup, "Stop
following " + m_model.m_strNickToFollow);
        }
        // if there's a selected name, which isn't me
        if (strNick.length() > 0 &&
!strNick.equals(m_model.getMyNick()))       // if it's
not me
        {
            m_miSendPrivate = addMenuHelper(popup,
"Send Private to " + strNick);
            m_miSendVerbol = addMenuHelper(popup, "Send
Verbol to " + strNick);
            popup.addSeparator();

            // specific verbol actions - shortcuts
            addMenuVerbolAction(popup, "Ask " + strNick
+ " to Follow Me", "FOLLOWME");
            addMenuVerbolAction(popup, "Ask " + strNick
+ " to Print", "PRINT");
            String strMyUrl =
m_model.getUsersTableModel().getUserLocation(strMyNick)
;
            if (strMyUrl != null)
            {
                addMenuVerbolAction(popup, "Ask " +
strNick + " to browse to my location",
                                                "BROWSE " +
m_model.getUsersTableModel().getUserLocation(strMyNick)
);
            }
            popup.addSeparator();
        }

        m_miRefreshUsers = addMenuHelper(popup,
"Refresh Users");
```

```
        //m_miFlipLayout = addMenuHelper(popup, "Flip
horizontal/vertical"); // removed - no-one uses it !
        m_miHistory = addMenuHelper(popup, "Show Older
Messages (History)...");

        if (m_nMode == MODE_IBOOK)
        {
            // in IBook mode, we can let hide the
window.
            // in General mode, we can't because, we
have NO way to let them open it again.
            m_miHideWindow = addMenuHelper(popup, "Hide
Window");
        }
        m_miOptions = addMenuHelper(popup,
"Options...");
        m_miChatIcons = addMenuHelper(popup, "Chat
Icons...");
        m_miClearWindow = addMenuHelper(popup, "Clear
Window");
        // m_miPrint = addMenuHelper(popup, "Print"); -
leave off for now
        return popup;
    }

/*******************************************************
     * Helper function to add a defined verbol action
onto the menu. There is the
     * possibility that this could be configureable at
some point in the future
     * @param menu Menu to add this item to
     * @param menuText Text to be displayed on the menu
     * @param verbolText Verbol message to be sent
```

```
*****************************************************/
    private JMenuItem addMenuVerbolAction(JPopupMenu
menu, String menuText, String verbolText)
    {
        JMenuItem mi = new JMenuItem(menuText);
        mi.setActionCommand("SEND * VERBOL! " +
verbolText); // who to send it too
        mi.addActionListener(this);     // so we listen
for menu commands
        menu.add(mi);
        return mi;
    }


/*****************************************************
     * Builds a popup menu with the names of everyone
in this chat room in

*****************************************************/
    private JPopupMenu buildSendPrivateMenu()
    {
        JPopupMenu popup = new JPopupMenu();
        Enumeration enum =
m_model.getUsersTableModel().getUsers();
        while (enum.hasMoreElements())
        {
            ChatModel.UserInfo user =
(ChatModel.UserInfo)enum.nextElement();
            String strNick = user.getNick();
            if (!strNick.equals(m_model.getMyNick()))
            {
                    JMenuItem mi = new JMenuItem("Send to "
+ user.getNick());
                    mi.setActionCommand("SEND " + strNick);
// who to send it too
                    mi.addActionListener(this);     // so
we listen for menu commands
                    popup.add(mi);
                }
            }
            return popup;
        }

        private final String getSelectedNick()
        {
            int nRow = m_tableUsers.getSelectedRow();
            if (nRow >= 0)
            {
                return
m_model.getUsersTableModel().getNickAt(nRow);   // get
the nickname
            }
            else
                return "";
        }


/*******************************************************
      * Sends the contents of the text box to the
selected invidual. If the text
      * box is empty tell the user they need to type
something.
      */
     private void onSendPrivateToSelected()
```

```
        {
            String strNick = getSelectedNick();
            onSendPrivateToNick(strNick);
        }

    /*****************************************************
    * Sends the contents of the text box to the specified
    individual. If the text
         * box is empty tell the user they need to type
    something.
         */
        private void onSendPrivateToNick(String strNick)
        {
            String strText = m_fieldInput.getText();
            if (strText.length() == 0)
            {
                JOptionPane.showMessageDialog(m_frame,
    "Please enter the message in the text entry box");
                return;
            }

            m_model.sendOutgoingMessage(strNick, strText);
            m_fieldInput.setText("");
        }


    /*****************************************************
         * Sends the contents of the text box to the
    selected invidual. If the text
         * box is empty tell the user they need to type
    something.
         */
        void onSendVerbolToSelected()
    {
        String strText = m_fieldInput.getText();
        if (strText.length() == 0)
        {
            JOptionPane.showMessageDialog(m_frame,
"Please enter the Verbol instruction in the text entry
box");
            return;
        }

        String strNick = getSelectedNick();
        m_model.sendOutgoingMessage(strNick, "VERBOL! "
+ strText);
        m_fieldInput.setText("");
    }

    void scrollToEnd()
    {
        try
        {
            if (USE_TEXTPANE)
                m_textChat.scrollToEnd();
            else

m_listChat.ensureIndexIsVisible(m_listChat.getModel().g
etSize() - 1);
        }
        catch (Exception e)
        {
            System.out.println("Exception handling
scrollToEnd()");
            e.printStackTrace();
        }
    }

/*******************************************************
* sends a NAMES message for this chat room - so the
user list effectively
    * refreshes on the reply

*******************************************************/
    void onRefreshUsers()
    {
        m_model.refreshUsers();
    }

/*******************************************************
     * Stop following.
     */
    void onFollowStop()
    {
        m_model.m_strNickToFollow = "";           //
cancel following
        m_model.getUsersTableModel().update();    // so
we update the display of the table
    }

/*******************************************************
     * Start following a specified nick - used by
Verbol.
     * This does NOT toggle the selection if we're
already following this person
     */
    private void onFollowSpecifiedNick(String strNick)
    {
        // check the nick is in the model.
```

```
            ChatModel.UserInfo user =
m_model.getUsersTableModel().getUserInfo(strNick);
            if (user == null)
            {
                JOptionPane.showMessageDialog(m_frame,
"Attempt to follow unknown user " + strNick);
                return;
            }

            // set a new nickname to follow, and browse to
where they are now
            m_model.m_strNickToFollow = strNick;
            String strLoc =
m_model.getUsersTableModel().getUserLocation(strNick);
            if (strLoc != null && strLoc.length() > 0)
                navigateTo(strLoc, true);

            m_model.getUsersTableModel().update();
// so we update the display of the table
        }

/*******************************************************
     * Sends the contents of the text box to the
selected invidual. If the text
     * box is empty tell the user they need to type
something.
     */
        private void onFollowSelectedNick()
        {
            String strNick = getSelectedNick();
            String strMyNick = m_model.getMyNick();
            if (strNick.equals(strMyNick))
            {
                JOptionPane.showMessageDialog(m_frame, "You
can't follow yourself");
                return;
            }
            if (m_model.m_strNickToFollow.equals(strNick))
                m_model.m_strNickToFollow = "";
// cancel following
            else
            {
                onFollowSpecifiedNick(strNick);
            }
            m_model.getUsersTableModel().update();
// so we update the display of the table
        }


/*****************************************************
     * Jump to this location (done once) - doesn't set
the follow flag.
     * Should it clear it ?
     */
        private void onGoThere()
        {
            String strNick = getSelectedNick();
            String strLoc =
m_model.getUsersTableModel().getUserLocation(strNick);
            navigateTo(strLoc, true);
        }


/*****************************************************
     * Hide the instance window, so we'll get those
fancy popups
```

```
*****************************************************/
    private void onHideWindow()
    {
        m_frame.setVisible(false);          // just hide
the frame - all other behaviour continues
        m_bShowFriendly = true; // show the friendly
messages
        System.out.println("Chat Frame hidden");
    }

    private void onOptions()
    {
        PropertySheet sheet = new
PropertySheet("Options");
        sheet.addPage("Chat", null, new
ChatPropertyPage());
        sheet.pack();
        sheet.setVisible(true);
    }

    private void onChatIcons()
    {
        if (m_chatIconFrame == null)
            m_chatIconFrame = new ChatIconFrame(this);

        m_chatIconFrame.setVisible(true);    //
otherwise
    }

    private void onClearWindow()
    {
        m_textChat.clear();
    }

    private void onPrint()
    {
        PrintJob job =
Toolkit.getDefaultToolkit().getPrintJob(m_frame,
"Print", null);

        Dimension sizePage = job.getPageDimension();
        Dimension sizeText = m_textChat.getSize();
        // This was an attempt to get printing working,
but the functionality in
        // JDK 1.1 is *VERY* limited. Left code in, but
it doesn't work properly

        int nPages = (int)Math.ceil(sizeText.height /
sizePage.height);

        // so we'have N pages...but no means of knowing
how many pages we've actually been asked to print !
        for (int i = 0; i < nPages; i++)
        {
            int y = (int)(i * sizePage.height);      //
offset for translating
            Graphics g = job.getGraphics();
            g.translate(0, y);
            m_textChat.printAll(g);
        }

        job.end();
    }
```

```
/*******************************************************
      * Implement the request for History.
      * Simply display a dialog allowing a number of
minutes to be entered.
      * Needs improvement for validation, and extending
the options, e.g.
      * show all messages today.
      *

*******************************************************/
    private void onHistory()
    {
        DlgHistory dlg = new DlgHistory();
        //dlg.pack();
        String[] ConnectOptionNames = { "OK", "Cancel"
};

        if (JOptionPane.showOptionDialog(m_frame, dlg,
"History",
           JOptionPane.PLAIN_MESSAGE,
JOptionPane.INFORMATION_MESSAGE,
                   null, ConnectOptionNames,
ConnectOptionNames[0]) == 0)
        {
            String strMsg = "HISTORY " +
m_model.getChannel().getName() + " "
                          + (dlg.getMinutes() * -60);

m_model.getChannel().getServerConnection().postRaw(strM
sg);
        }
        }

        public void executeVerbol(String strFrom, String
    strVerbol, boolean bExecuteWithoutAsking)
        {
            if (!bExecuteWithoutAsking &&
    !m_ext.isUserTrusted(strFrom))
            {
                AskExecuteVerbolPanel panel = new
    AskExecuteVerbolPanel(strFrom, strVerbol);
                int nOption =
    JOptionPane.showConfirmDialog(m_frame, panel,
                        "Verbol From " + strFrom,
                        JOptionPane.YES_NO_OPTION);

                if (nOption != JOptionPane.YES_OPTION)
                    return;         // didn't click yes, so
    don't do it

                if (panel.acceptNextTime())
                    m_ext.addTrustedVerbolUser(strFrom);
            }

            // have to implement FOLLOWME locally in the
    chat controller - doesn't go anywhere near the ibook.
            // This is basically a quick fix to allow Sandy
    to print documents for Brian.
            StringTokenizer tok = new
    StringTokenizer(strVerbol, " ");
            String strVerb = tok.nextToken();
            if (strVerb.equalsIgnoreCase("FOLLOWME"))
            {
                onFollowSpecifiedNick(strFrom);
                return;
            }
            executeVerbol(strVerbol);
        }


    /*****************************************************
         * Inserts the specified text into the entry field
    for this chat window.
         * Called from the ChatIconFrame.

    *****************************************************/
        public void insertEntryFieldText(String strText)
        {
            m_fieldInput.replaceSelection(strText);
        }

        //
    *****************************************************
        // listener for mouse clicks on the table.
        //
    *****************************************************
        class PopupListener extends MouseAdapter
        {
            public void mouseClicked(MouseEvent e)
            {
                if (e.getSource() == m_textChat) // on
    single click see if we can browse to the URL
                {
                    Point pt = e.getPoint();
                    String strText =
    m_textChat.getTextAtPoint(pt);
                    if (ChatTextPane.isURL(strText))
                    {
                        navigateTo(strText, true);
                    }
                }
                else if (e.getSource() == m_tableUsers &&
e.getClickCount() == 2)
                {
                    onGoThere();
                }

            }

            public void mousePressed(MouseEvent e)
            {
                maybeShowPopup(e);
            }

            public void mouseReleased(MouseEvent e)
            {
                maybeShowPopup(e);
            }

            private void maybeShowPopup(MouseEvent e)
            {
                if (e.getSource() == m_tableUsers &&
e.isPopupTrigger())
                {
                    // for testing - see where events are
happening....
                    System.out.println("Chat.popup Thread="
+ Thread.currentThread().getName());
```

```
                    int nRow =
m_tableUsers.rowAtPoint(e.getPoint());
                    ListSelectionModel sel =
m_tableUsers.getSelectionModel();
                    sel.setSelectionInterval(nRow, nRow);
                    JPopupMenu menu = buildPopupMenu();
                    menu.show(e.getComponent(),
                                   e.getX(), e.getY());
               }
          }
     }



     //
************************************************************************
     // Special renderer for rendering the Nick column
(draws an icon next to the
     // user if this is the person we are following).
Based on ColorRenderer class
     // in JFC tutorial
     //
************************************************************************
     class NickRenderer extends JLabel
                         implements TableCellRenderer
     {
          Border unselectedBorder = null;
          Border selectedBorder = null;
          boolean isBordered = true;
          private ImageIcon   m_iconArrow;

          public NickRenderer(boolean isBordered)
          {
               super();
```

```
            this.isBordered = isBordered;
            m_iconArrow =
loadImageIcon("/images/follow.gif");
            setOpaque(true); //MUST do this for
background to show up.
        }

        public Component getTableCellRendererComponent(
                                JTable table, Object
color,
                                boolean isSelected,
boolean hasFocus,
                                int row, int column)
        {
            try
            {
                ChatModel.UserTableModel model =
(ChatModel.UserTableModel)table.getModel();
                ChatModel.UserInfo user =
model.getUserInfo(row);
                String strNick = user.getNick();
                if
(strNick.equals(model.getNickToFollow()))
                {
                    setIcon(m_iconArrow);
                }
                else
                    setIcon(null);
                setFont(table.getFont());
                setText(user.getDisplayName());
                setBackground(isSelected ?
table.getSelectionBackground() :
table.getBackground());
```

```
                    if (isBordered) {
                        if (isSelected) {
                            if (selectedBorder == null) {
                                selectedBorder =
BorderFactory.createMatteBorder(2,5,2,5,

table.getSelectionBackground());
                            }
                            setBorder(selectedBorder);
                        } else {
                            if (unselectedBorder == null) {
                                unselectedBorder =
BorderFactory.createMatteBorder(2,5,2,5,

table.getBackground());
                            }
                            setBorder(unselectedBorder);
                        }
                    }
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
                return this;
            }
        }



        public void intervalAdded(ListDataEvent e)
        {
            // attempt to ensure bottom of list is visible
            // This doesn't seem to work reliably in the
    Borland JVM, though seems to be
            // better in the IBM JDK.
            // Have tried various different approches
    including invalidating the list
            // each time, but to no avail. The
    invokeLater() ensures that the list is
            // always scrolled from the AWT Event thread.
            scrollToEnd();
            /*
            SwingUtilities.invokeLater(new Runnable()
            {
                public void run()
                {
                    scrollToEnd();
                }
            });*/
        }

        public void intervalRemoved(ListDataEvent e)
        {
            // do nothing
        }
        public void contentsChanged(ListDataEvent e)
        {
            // do nothing
        }

    /******************************************************
         * Loads a image icon given the resource string,
    reporting if it;s not found
```

```
*****************************************************/
    public ImageIcon loadImageIcon(String strResource)
    {
        URL url = getClass().getResource(strResource);
        if (url == null)
        {
            System.out.println("Couldn't find " +
strResource);
            return new ImageIcon();
        }
        else
            return new ImageIcon(url);
    }


/*****************************************************
     * Dump debug information
     */
    private void dumpDebug()
    {
        System.out.println(m_model.toString());
        ServerConnection.dumpInfo();
    }

/*****************************************************
     * Open IE looking at the specified file. Method
here is to just open HTML
     * files using cmd.exe - seems unnecessary to
actually go looking for the
     * location of IE to launch the correct process.
Technically, if another
```

```
        * browser is registered for HTML this will be a
problem though
        */
    static public void launchHelp(String strFile)
    {
        String url = System.getProperty("user.dir") +
"\\interface\\docs\\controller\\xh\\" + strFile;
        System.out.println("open help at " + url);
        try
        {
            BrowserLauncher.openURL(url);
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }

        /*
        String[] cmdargs = {"cmd.exe", "/c",
System.getProperty("user.dir") +
"\\..\\docs\\controller\\xh\\" + strFile};
        Runtime runtime = Runtime.getRuntime();
        try
        {
            System.out.println("opening help at " +
cmdargs[2]);
            runtime.exec(cmdargs);
        }*/
    }
}
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatclient/Channel.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++
package com.fs.chatclient;

import java.util.*;
/**
 * Represents a Channel on a chat server. Obtained by
calling ServerConnection.joinChannel()
 * user can send messages to this Channel, by calling
send message, etc.
 *
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */

public class Channel implements ChatListener
{
    String m_strName;        // name of the channel -
includes the leading #
    String m_strTopic;       // channel topic
    Vector m_vecNicks = new Vector();       // nicks
currently in this channel
    private ServerConnection    m_server;

    Channel(ServerConnection server, String strChannel)
    {
        m_server = server;
        m_strName = strChannel;
    }
    public String toString()
    {
        return (super.toString() + " (channel=" +
m_strName + " connection=" + m_server.toString() +
")");
    }

    public String getTopic()
    {
        return m_strTopic;
    }

    public String getName()
    {
        return m_strName;
    }

    public ServerConnection getServerConnection()
    {
        return m_server;
    }

    public void serverMessage(int nCode, String strMsg)
    {
    }


    public boolean chatEvent(ChatMessage msg)
    {
        int nCode = msg.getResponseCode();
        switch (nCode)
        {
                case Codes.RPL_NAMREPLY:
                {
                    if (m_strName.equals(msg.getParam(0)))
                    {
                        m_vecNicks.removeAllElements();
                        StringTokenizer tok = new
StringTokenizer(msg.getLastParam(), " ");
                        while (tok.hasMoreElements())
                        {
                            String strName =
tok.nextToken();
                            addName(strName);
                        }
                    }
                    return true;
                }

                case Codes.RPL_TOPIC:
                    if (m_strName.equals(msg.getParam(0)))
                    {
                        m_strTopic = msg.getLastParam();
                        return true;
                    }
            }

            return false;

        }

        public void postMessage(String strText)
        {
            String strOut = "PRIVMSG " + m_strName + " :" +
strText;
```

```
            m_server.postRaw(strOut);
        }

    /*****************************************************
         * A clean exit from the Channel.
         * Sends a PART message and removes the Channel
    from the server, so we don't
         * re-join it.
         * @param strQuitMessage Message to send when the
    user "leaves"
         */
        public void closeChannel(String strQuitMessage)
        {
            // send a PART message to the channel
            // if the channel was "held open"
            String strOut = "PART " + m_strName;
            if (strQuitMessage != null)
                strOut += " :" + strQuitMessage;

            getServerConnection().postRaw(strOut);
            m_server.removeChannel(this);                //
    remove this channel from the server
        }


    /*****************************************************
         * Adds a name to the list of names in the chat
    room
         */
        private void addName(String strName)
        {
            m_vecNicks.addElement(strName);
        }
    }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatclient/ChatListener
.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++
package com.fs.chatclient;

import java.util.Vector;
/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */

public interface ChatListener
{

/*****************************************************
******************
     * Notification of an incoming chat message. Return
true if the message was
     * handled or is NOT relevant to other listeners
(eg other channels)
     */
    public boolean chatEvent(ChatMessage msg);
    public void serverMessage(int nCode, String
strMsg);       // server message (e.g. Connecting....,
Connection FAILED, etc).
}
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatclient/ChatMessage.
java
//+++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatclient;

/**
* $Date: 2002/04/15 22:25:12 $
* $Author: markc $
* $Revision: 1.4 $
*
* @author Mark Conway
*
* NB: this class was "hijacked" from the client side.
Originally intended there to
* be a shared package for the client and server side.
*
* This class is responsible for parsing a Chat message
- as specified by RFC1459.
* Could probably be extended to build up messages, but
this is pretty easy to do just by contructing
* the text itself.
*
* Basic format for a message is:
* [:prefix-area] commands (...) [:message]
*
* Prefix area is indicated by a colon at the start of
the message (must be 1st character)
* 1 or more spaces are regarded as separators.
* A command plus 0-15 parameters may be present
* The final parameter may optionally begin with a
colon, which indicates the remainder
```

```
* of the string is a message (ie spaces are
significant)
*
* If the reply is numeric (3-digit code), then record
the numeric code
*/

import java.util.*;
import java.text.SimpleDateFormat;

public class ChatMessage
{
    static public final String STANDARD_DATE_FORMAT =
"yyyyMMdd-HHmmss";
    static public final int PROTOCOL_VERSION = 2;


    private String  m_strPrefix;   // prefix (optional)
    private Vector  m_vecParams = new Vector();   //
vector of commands (0-15)
    private int           m_nCode;        // numeric
reply code (0 if item is NOT numeric)
    private Date    m_time = null;          // time of
the message - if extracted by the time string -
otherwise none.
    private boolean m_bColonIncluded = false;

    static private SimpleDateFormat g_dateFormat;

    static
    {
        // initializate date format used to parse
incoming dates from server (which are in GMT)
```

```
            g_dateFormat = new
    SimpleDateFormat(STANDARD_DATE_FORMAT);

    g_dateFormat.setTimeZone(TimeZone.getTimeZone("GMT"));
        }



        public ChatMessage()
        {
        }

        /**
         * main() method just used to simplify testing.
         */
        public static void main (String args[])
        {
            // just used to test.
            ChatMessage msg = parseString(":blah@x.com
    NOTICE :Hello World !");
            System.out.println(msg);

            msg = parseString(":mrchome 451 :you are not
    registered on the server");
            System.out.println(msg);


            msg = parseString(":test 311 markc :Mark
    Conway");
            assertEquals(msg.getFreeFormParam(), "Mark
    Conway");
            System.out.println(msg);
    }

    private static void assertEquals(Object o1, Object
o2)
    {
        if (!o1.equals(o2))
            System.out.println("fail - " + o1 + " is
not equal to " + o2);
    }


/*******************************************************
*******************
     * static method returning the parsed message

*******************************************************
*****************/
    static public ChatMessage parseString (String str)
    {
        ChatMessage msg = new ChatMessage();
        msg.parse(str);
        return msg;
    }


/*******************************************************
*******************
     * returns the prefix for the message
     * @return message prefix (sender of the message)

*******************************************************
*****************/
    public String getPrefix()
```

```
        {
            return m_strPrefix;
        }


/*******************************************************
*******************
      * returns the last parameter - usually the text
message
      * this is a convenience method as this is often
the most important part of
      * the message.
      * @return last parameter of the message

*******************************************************
******************/
    public String getLastParam()
    {
         return
(String)m_vecParams.elementAt(m_vecParams.size() - 1);
    }


/*******************************************************
*******************
      * returns the last parameter - but ONLY if it was
specified via a Colon - otherwise
      * returns null
      * @return last parameter of the message

*******************************************************
******************/
```

```
    public String getFreeFormParam()
    {
        if (m_bColonIncluded)
            return
(String)m_vecParams.elementAt(m_vecParams.size() - 1);
        else
            return null;
    }



/*******************************************************
*******************
     * returns the parameter at that number.
     * @param nIndex Nth parameter to return. 0 = main
command
     * @return parameter

********************************************************
******************/
    public String getParam(int nIndex)
    {
        if (nIndex < m_vecParams.size())
        return (String)m_vecParams.elementAt(nIndex);
        else
            return "";
    }

    public int getParameterCount()
    {
        return m_vecParams.size();
    }
```

```
/*****************************************************
******************
     * @return the numeric response code (0 if not a
numeric response)

*****************************************************
******************/
    public int getResponseCode()
    {
        return m_nCode;
    }


/*****************************************************
******************
     * @return the Nickname or idenitification of the
contributor of a message -
     * not always valid - typically just for PRIVMSG,
etc

*****************************************************
******************/
    public String getContributor()
    {
        String strPrefix = getPrefix();
        if (strPrefix == null)
            return null;           // no prefix --> no
contributor
        int nPling = strPrefix.indexOf('!');
        if (nPling != -1)
            return strPrefix.substring(0, nPling);
        else
```

```
            return strPrefix;       // go for the entire
prefix.
    }

/*******************************************************
******************
     * return a string representation of the parameters
(NOT the actual message !).
*******************************************************
*****************/
    public String toString()
    {
        StringBuffer buf = new StringBuffer();
        buf.append("prefix=");
        buf.append(m_strPrefix);
        for (int i = 0; i < m_vecParams.size(); i++)
        {
            buf.append(",");
            buf.append(m_vecParams.elementAt(i));
        }
        buf.append(",code=");
        buf.append(m_nCode);
        return buf.toString();
    }

/*******************************************************
******************
     * Parses the raw chat message and returns a
string.
     * @param strIn incoming IRC message to parse
```

```
******************************************************
*****************/
    public void parse(String strIn)
    {
        if (strIn == null || strIn.length() == 0)
            return;             // cope with empty
strings

        int nSt = 0;
        int nEnd;

        // if first char is ":" then we have a prefix
area which finishes at the first space.
        // otherwise get parameter to first space. if
begins with ":", it's the rest of the line.
        if (strIn.charAt(0) == ':')
        {
            nEnd = safeIndexOf(strIn, ' ', nSt);
            m_strPrefix = strIn.substring(1, nEnd);
     // skip leading colon
            nSt = nEnd;
        }
        nSt = skipSpaces(strIn, nSt);

        while (nSt < strIn.length())
        {
            if (strIn.charAt(nSt) == ':')
            {
                m_bColonIncluded = true;

m_vecParams.addElement(strIn.substring(nSt + 1));
                break;
                }
                nEnd = safeIndexOf(strIn, ' ', nSt);
                String strTok = strIn.substring(nSt, nEnd);
                m_vecParams.addElement(strTok);
                nSt = skipSpaces(strIn, nEnd);
            }

            // see if the first string (the command) was a
numeric - ie a response code.
            if (m_vecParams.size() > 0)
            {
                String strCmd =
(String)m_vecParams.elementAt(0);
                try
                {
                    m_nCode = Integer.parseInt(strCmd);
                }
                catch (NumberFormatException e)
                {
                    m_nCode = 0;  // not a valid numeric
code
                }
            }
        }


/*******************************************************
******************
     * Helper function: returns length of string
instead of -1, if not found
```

```
*******************************************************
*****************/
    static private int safeIndexOf(String str, int nCh,
int nPos)
    {
        int nRet = str.indexOf(nCh, nPos);
        return (nRet == -1 ? str.length()  : nRet);
    }


/******************************************************
******************
     * Helper Function: returns next non-space
character - stopping at end of string

*******************************************************
*****************/
    static private int skipSpaces(String str, int nPos)
    {
        while (nPos < str.length() && str.charAt(nPos)
== ' ')
            nPos++;
        return nPos;
    }


/******************************************************
******************
     * This method extracts the GMT format time from
the message - embedded
     * in [,] and removes it from the LastParameter.
Intended to be called
```

```
     * for messages in which a time may be embedded i.e
PRIVMSG, NOTICE

*******************************************************
****************/
     public boolean parseTime()
     {
        if (m_time != null)     // already extracted a
date previously
            return true;

        String strMsg = getLastParam();
        int nBrace = 0;
        if (!strMsg.startsWith("[") || (nBrace =
strMsg.indexOf(']')) < 0 )
            return false;          // no time - don't
touch the message !

        String strTime = strMsg.substring(1,
nBrace).trim();
        try
        {
            m_time = g_dateFormat.parse(strTime); //
should have Date as parsed from GMT supplied by Server.
        }
        catch (java.text.ParseException e)
        {
            return false;       // no time !
            // invalid date - should return
        }
        if (nBrace + 1 < strMsg.length())
            strMsg = strMsg.substring(nBrace +
1).trim();   // end of string - remove any whitespacec:
```

```
            else
                strMsg = "";
            m_vecParams.setElementAt(strMsg,
m_vecParams.size() - 1);
            return true;            // it had a time !
        }



/*******************************************************
*******************
     * Returns the time embedded in the message as a
string. parseTime() must
     * have been explicitly called on the message to
extract the time field

*******************************************************
****************/
     public String getDateString()
     {
          String strTime = "";
          if (m_time != null)
          {
               Calendar calMsg = Calendar.getInstance();
// calendar for my timezone.
               calMsg.setTime(m_time);

               Calendar calNow = Calendar.getInstance();
               calNow.setTime(new Date());


               // assume YEAR is really unlikely to
matter....
```

```
            // if it's TODAY, the format using just the
time
            if (calMsg.get(Calendar.DAY_OF_YEAR) ==
calNow.get(Calendar.DAY_OF_YEAR))
            {
                SimpleDateFormat fmt = new
SimpleDateFormat("HH:mm");
                strTime = fmt.format(m_time);
            }
            else
            {
                SimpleDateFormat fmt = new
SimpleDateFormat("MMM dd,HH:mm");
                strTime = fmt.format(m_time);
            }
        }
        return strTime;
    }


/*******************************************************
********************
     * Returns the timestamp for the message, or null
if none
     */
     public Date getTimestamp()
     {
        return m_time;
     }

}
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatclient/ChatPane.jav
a
//++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatclient;

// This class isn't used
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

import java.util.*;

//SplitPaneDemo itself is not a visible component.
public class ChatPane implements ChatListener,
ActionListener
{
    private JSplitPane m_paneSplit;     // splitter
pane
    private JPanel m_paneMain;          // this panel
    private JTextField m_fieldInput;    // input text
area
    private JTable m_tableUsers;        // user list
    private JTextArea m_textChat;       // text area
for chat messages

    private Channel m_channel;          // channel to
use

    public ChatPane()
    {
        m_paneMain = new JPanel();
        m_paneMain.setLayout(new BorderLayout());

        // create the split pane and put this in the
border layour with an entry field.
        createSplitPane();

        m_fieldInput = new JTextField(40);
        m_fieldInput.setText("type stuff here");
        //m_fieldInput.addKeyListener(new
EnterKeyListener());
        m_fieldInput.addActionListener(this);

        m_paneMain.add(m_fieldInput,
BorderLayout.SOUTH);
        m_paneMain.add(m_paneSplit,
BorderLayout.CENTER);
    }


/*****************************************************
*****************
     * Specify the channel that this UI should work
with. It adds itself as a listener
     */
    public void setChannel(Channel channel)
    {
        if (m_channel == null)
        {
            m_channel = channel;

channel.getServerConnection().addListener(this);
```

```
        }
    }


/*****************************************************************************
     * Handle action events from the text input field -
basically sends a message
     */
    public void actionPerformed(ActionEvent evt)
    {
        String text = m_fieldInput.getText();
        m_channel.postMessage(text);            // post
the message to the server

appendMessage(m_channel.getServerConnection().getNick()
, text); // update our display
        m_fieldInput.setText("");        // clear the
text
    }

    public void appendMessage(String strFrom, String
strText)
    {
        m_textChat.append(strFrom + " > " + strText +
"\n");

    }


/*****************************************************************************
     * implement the ChatListener interface
     */
    public boolean chatEvent(ChatMessage msg)
    {
        String strCommand = msg.getParam(0);
        if (strCommand.equals("PRIVMSG") ||
strCommand.equals("NOTICE"))
        {
            if
(m_channel.getName().equals(msg.getParam(1)))
            {   // message for this channel
                appendMessage(msg.getContributor(),
msg.getLastParam());

            }

        }
        return false;
    }

    public void usersUpdated(Vector vecUsers)
    {
    }



    public void serverMessage(int nCode, String strMsg)
    {
        appendMessage("[SERVER]", strMsg);
    }



    private void createSplitPane()
    {
        m_textChat = new JTextArea(5, 20);
        m_textChat.setEditable(false);
        JScrollPane textScrollPane = new
JScrollPane(m_textChat,

JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,

JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);


        // create a table to go in the right pane.
        String[] arrColNames = { "user", "at" };

        Object[][] data = {
                        { "Brian", "www.ibook.com"} ,
                        { "Charly",
"www.shalbournetech.com" }
        };

        m_tableUsers = new JTable(data, arrColNames);
        m_tableUsers.setShowHorizontalLines(false);
        m_tableUsers.setShowVerticalLines(false);
        JScrollPane tableScrollPane = new
JScrollPane(m_tableUsers);

        //Create a split pane with the two scroll panes
in it
        m_paneSplit = new
JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                                    textScrollPane,
tableScrollPane);
        m_paneSplit.setOneTouchExpandable(true);
        m_paneSplit.setDividerLocation(150);
```

```
            //Provide minimum sizes for the two components
    in the split pane
            Dimension minimumSize = new Dimension(100, 50);
            textScrollPane.setMinimumSize(minimumSize);
            tableScrollPane.setMinimumSize(minimumSize);

            //Provide a preferred size for the split pane
            m_paneSplit.setPreferredSize(new Dimension(400,
    200));
        }

        public JPanel getPanel()
        {
            return m_paneMain;
        }


        public static void main(String s[])
        {
            JFrame frame = new JFrame("ChatPane");

            frame.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e)
    {System.exit(0);}
            });


            ChatPane ChatPane = new ChatPane();

    frame.getContentPane().add(ChatPane.getPanel());
            frame.pack();
            frame.setVisible(true);
        }
    }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatclient/Codes.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatclient;

/**
 * Symbolic constants used in the RFC
 * $Date: 2002/02/19 13:47:36 $
 * $Author: markc $
 * $Revision: 1.1 $
 * @author Mark Conway
 * @version 1.0
 */

public final class Codes
{
    static public final int   RPL_TRACELINK   =   200
     ;
    static public final int   RPL_TRACECONNECTING =
     201  ;
    static public final int   RPL_TRACEHANDSHAKE  =
     202  ;
    static public final int   RPL_TRACEUNKNOWN    =
     203  ;
    static public final int   RPL_TRACEOPERATOR   =
     204  ;
    static public final int   RPL_TRACEUSER   =   205
     ;
    static public final int   RPL_TRACESERVER     =
     206  ;
    static public final int   RPL_TRACENEWTYPE    =
     208  ;
```

```
static public final int  RPL_TRACECLASS =  209
;
static public final int  RPL_STATSLINKINFO  =
211  ;
static public final int  RPL_STATSCOMMANDS  =
212  ;
static public final int  RPL_STATSCLINE =  213
;
static public final int  RPL_STATSNLINE =  214
;
static public final int  RPL_STATSILINE =  215
;
static public final int  RPL_STATSKLINE =  216
;
static public final int  RPL_STATSYLINE =  218
;
static public final int  RPL_ENDOFSTATS =  219
;
static public final int  RPL_UMODEIS    =  221
;
static public final int  RPL_SERVICEINFO    =
231  ;
static public final int  RPL_SERVICE    =  233
;
static public final int  RPL_SERVLISTEND    =
235  ;
static public final int  RPL_STATSLLINE =  241
;
static public final int  RPL_STATSUPTIME    =
242  ;
static public final int  RPL_STATSOLINE =  243
;
```

```
    static public final int   RPL_STATSHLINE =    244
     ;
    static public final int   RPL_LUSERCLIENT        =
     251  ;
    static public final int   RPL_LUSEROP     =    252
     ;
    static public final int   RPL_LUSERUNKNOWN       =
     253  ;
    static public final int   RPL_LUSERCHANNELS      =
     254  ;
    static public final int   RPL_LUSERME      =    255
     ;
    static public final int   RPL_ADMINME      =    256
     ;
    static public final int   RPL_ADMINLOC1    =    257
     ;
    static public final int   RPL_ADMINLOC2    =    258
     ;
    static public final int   RPL_ADMINEMAIL =    259
     ;
    static public final int   RPL_TRACELOG     =    261
     ;
    static public final int   RPL_NONE   =    300  ;
    static public final int   RPL_AWAY   =    301  ;
    static public final int   RPL_USERHOST     =    302
     ;
    static public final int   RPL_ISON   =    303  ;
    static public final int   RPL_UNAWAY       =    305
     ;
    static public final int   RPL_NOWAWAY      =    306
     ;
    static public final int   RPL_WHOISUSER    =    311
     ;
```

```
static public final int  RPL_WHOISSERVER     =
 312  ;
static public final int  RPL_WHOISOPERATOR   =
 313  ;
static public final int  RPL_WHOWASUSER =    314
 ;
static public final int  RPL_ENDOFWHO    =   315
 ;
static public final int  RPL_WHOISCHANOP     =
 316  ;
static public final int  RPL_WHOISIDLE  =    317
 ;
static public final int  RPL_ENDOFWHOIS =    318
 ;
static public final int  RPL_WHOISCHANNELS   =
 319  ;
static public final int  RPL_LISTSTART  =    321
 ;
static public final int  RPL_LIST  =    322  ;
static public final int  RPL_LISTEND    =    323
 ;
static public final int  RPL_CHANNELMODEIS   =
 324  ;
static public final int  RPL_NOTOPIC    =    331
 ;
static public final int  RPL_TOPIC =    332  ;
static public final int  RPL_INVITING   =    341
 ;
static public final int  RPL_SUMMONING  =    342
 ;
static public final int  RPL_VERSION    =    351
 ;
```

```
static public final int    RPL_WHOREPLY    =    352
 ;
static public final int    RPL_NAMREPLY    =    353
 ;
static public final int    RPL_CLOSING     =    362
 ;
static public final int    RPL_LINKS =    364   ;
static public final int    RPL_ENDOFLINKS =    365
 ;
static public final int    RPL_ENDOFNAMES =    366
 ;
static public final int    RPL_BANLIST     =    367
 ;
static public final int    RPL_ENDOFBANLIST     =
 368  ;
static public final int    RPL_ENDOFWHOWAS      =
 369  ;
static public final int    RPL_INFO   =    371   ;
static public final int    RPL_MOTD   =    372   ;
static public final int    RPL_INFOSTART   =    373
 ;
static public final int    RPL_ENDOFINFO   =    374
 ;
static public final int    RPL_MOTDSTART   =    375
 ;
static public final int    RPL_ENDOFMOTD   =    376
 ;
static public final int    RPL_YOUREOPER   =    381
 ;
static public final int    RPL_REHASHING   =    382
 ;
static public final int    RPL_TIME   =    391   ;
```

```
        static public final int   RPL_USERSSTART =    392
         ;
        static public final int   RPL_USERS =    393  ;
        static public final int   RPL_ENDOFUSERS =    394
         ;
        static public final int   RPL_NOUSERS     =    395
         ;
        static public final int ERROR_START = 400;       //
    first error code - anything < 400 is NOT an error
        static public final int   ERR_NOSUCHNICK =    401
         ;
        static public final int   ERR_NOSUCHSERVER     =
         402  ;
        static public final int   ERR_NOSUCHCHANNEL    =
         403  ;
        static public final int   ERR_CANNOTSENDTOCHAN
         =    404  ;
        static public final int   ERR_TOOMANYCHANNELS =
         405  ;
        static public final int   ERR_WASNOSUCHNICK    =
         406  ;
        static public final int   ERR_TOOMANYTARGETS   =
         407  ;
        static public final int   ERR_NOORIGIN    =    409
         ;
        static public final int   ERR_NORECIPIENT      =
         411  ;
        static public final int   ERR_NOTEXTTOSEND     =
         412  ;
        static public final int   ERR_NOTOPLEVEL =    413
         ;
        static public final int   ERR_WILDTOPLEVEL     =
         414  ;
```

```
static public final int  ERR_UNKNOWNCOMMAND  =
 421  ;
static public final int  ERR_NOMOTD     =    422
 ;
static public final int  ERR_NOADMININFO     =
 423  ;
static public final int  ERR_FILEERROR  =    424
 ;
static public final int  ERR_NONICKNAMEGIVEN =
 431  ;
static public final int  ERR_ERRONEUSNICKNAME
 =    432  ;
static public final int  ERR_NICKNAMEINUSE   =
 433  ;
static public final int  ERR_NICKCOLLISION   =
 436  ;
static public final int  ERR_USERNOTINCHANNEL
 =    441  ;
static public final int  ERR_NOTONCHANNEL    =
 442  ;
static public final int  ERR_USERONCHANNEL   =
 443  ;
static public final int  ERR_NOLOGIN    =    444
 ;
static public final int  ERR_SUMMONDISABLED  =
 445  ;
static public final int  ERR_USERSDISABLED   =
 446  ;
static public final int  ERR_NOTREGISTERED   =
 451  ;
static public final int  ERR_NEEDMOREPARAMS  =
 461  ;
```

```
static public final int  ERR_ALREADYREGISTRED
 =    462  ;
static public final int  ERR_NOPERMFORHOST   =
 463  ;
static public final int  ERR_PASSWDMISMATCH  =
 464  ;
static public final int  ERR_YOUREBANNEDCREEP
 =    465  ;
static public final int  ERR_YOUWILLBEBANNED =
 466  ;
static public final int  ERR_KEYSET      =    467
 ;
static public final int  ERR_CHANNELISFULL   =
 471  ;
static public final int  ERR_UNKNOWNMODE     =
 472  ;
static public final int  ERR_INVITEONLYCHAN  =
 473  ;
static public final int  ERR_BANNEDFROMCHAN  =
 474  ;
static public final int  ERR_BADCHANNELKEY   =
 475  ;
static public final int  ERR_NOPRIVILEGES    =
 481  ;
static public final int  ERR_CHANOPRIVSNEEDED
 =    482  ;
static public final int  ERR_CANTKILLSERVER  =
 483  ;
static public final int  ERR_NOOPERHOST =    491
 ;
static public final int  ERR_NOSERVICEHOST   =
 492  ;
```

```
    static public final int   ERR_UMODEUNKNOWNFLAG
     =    501  ;
    static public final int   ERR_USERSDONTMATCH  =
     502  ;
}
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatclient/Queue.java
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatclient;

import java.util.Vector;

/**
 * Implementation of a simple Queue.
 *
 * Has to run on JDK 1.1 - so done in terms of Vector
rather than a linked list, etc.
 * $Date: 2002/02/19 13:47:36 $
 * $Author: markc $
 * $Revision: 1.1 $
 */

public class Queue
{
    Vector m_vec = new Vector();

    public Queue()
    {
    }


/*************************************************************
**************
     * Returns true if the queue is empty (has no items
in it)
     */
   public synchronized boolean isEmpty()
```

```
        {
            return (m_vec.size() == 0);
        }


    /****************************************************
    **************
         * Push an object onto the tail of the queue
         */
        public synchronized void push(Object obj)
        {
            m_vec.addElement(obj);
        }


    /****************************************************
    **************
         * Pop object off the head of the queue
         */
        public synchronized Object pop()
        {
            // not an efficient implementation - means we
    move the elements down the
            // array ! Replace with LinkedList in JDK 1.2
            Object obj = m_vec.elementAt(0);
            m_vec.removeElementAt(0);
            return obj;
        }


    /****************************************************
    **************
```

```
     * Push an object onto the head of the queue.
 Typically used to push
     * an object back onto the head of the queue, when
 it's just been popped off.
     */
    public synchronized void pushHead(Object obj)
    {
        m_vec.insertElementAt(obj, 0);
    }

    }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatclient/ServerConnec
tion.java
//+++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatclient;

/**
 * implements connection to a Server.
 * An instance of this class represents a connection to
a server. The lifespan of
 * these instances are reference counted, and a
connection may be shared by multiple
 * clients. E.g. logons to different channels on the
same server.
 * @author
 * @version $Revision: 1.12 $ $Date: 2002/04/20
22:36:22 $
 * $Author: markc $
 */
import java.util.*;
import java.net.*;
import java.io.*;

public class ServerConnection
{
    private static final String CVS_VERSION = "$Name:
$";   // filled in by CVS export

    private String m_strUser;
    private String m_strNick;
    private String m_strOriginalNick;   // nick name
originally specified
```

```
        private String m_strPass;
        private String m_strServer;
        private String m_strHost = null;     // host to use
    for connections. can be specified,
        private int     m_nPort;
        private String m_strRealName;         // real user
    name.

        private BufferedReader m_receive = null;     //
    incoming stream
        private boolean m_bStopped = false;      // set to
    true to indicate to threads that this connection is
    closing

        private Queue m_queue = new Queue();
    // should use a linked list, but unfortunately need to
    run in JDK 1.1
        private ServerConnectionThread m_threadSender;  //
    thread used to connect/send
        private Thread m_threadReceiver;                //
    thread read input from the socket
        private Vector  m_vecListeners = new Vector();  //
    listeners interested in the chat protocol.
        private Vector  m_vecChannels = new Vector();   //
    channels connected to (as Strings) - on re-connect we
    auto-join these channels !

        private Object m_logonSemaphore = new Object();
    // logon semaphore - used to communicate between
    Sender/receiver thread

    // when waiting for a login response
```

```
        private int     m_nRefCount = 0;
    // reference count, added whenever this is handed out
    to a client
    // when reaches zero, connection can be closed for
    real.
        // connection states
        public final int STATE_NO_SOCKET = 0;
    // socket NOT (yet)
        public final int STATE_AWAITING_LOGON_RESPONSE = 1;
    // socket open - waiting for logon response
        public final int STATE_ACTIVE = 2;
    // authorization appears to be succesful
        public final int STATE_CONNECTION_FAILED = 3;
    // repeated failure to connect - likely a
    network/configuration problem.
        private int m_nConnectState = STATE_NO_SOCKET;
        private int m_nConnectErrorCode = 0;
    // Error code originating from failure to connect

        static private Vector g_ServerConnections = new
    Vector();

    /*******************************************************
    ********************
        * Factory method for obtaining a server
    connection. Using this in preference
        * to a public constructor allows ServerConnections
    to be re-used if the same same,
        * port, etc is set up.

    *******************************************************
    ******************/
```

```
        static public String getCVSVersion()
        {
            return CVS_VERSION.substring(5);
        }


    /*****************************************************
    ******************
        * Factory method for obtaining a server
    connection.
        * This method will return a ServerConnection that
    is already in use. This is
        * necessary because a single chat server will
    typically only allow the same NICK
        * to be registered once. If a connection is
    shared, a reference count is used to
        * track the number of users of the connection.
    Only when this count reaches
        * zero is the underlying physical connection
    actually closed.

    *****************************************************
    ****************/
        static public ServerConnection
    getServerConnection(String strServer, int nPort, String
    strUser, String strNick, String strPass)
        {
            ServerConnectionInfo info = new
    ServerConnectionInfo();
            info.m_strServer = strServer;
            info.m_nPort = nPort;
            info.m_strUser = strUser;
            info.m_strNick = strNick;
            info.m_strPass = strPass;
            return getServerConnection(info);
        }



/***********************************************************************
*******************
        * Factory method for obtaining a server
connection. Using this in preference
        * to a public constructor allows ServerConnections
to be re-used if the same same,
        * port, etc is set up.

************************************************************************
*****************/
        static public ServerConnection
getServerConnection(ServerConnectionInfo info)
        {
            // find a suitable server connection.
            ServerConnection conn =
findServerConnection(info);
            if (conn == null) // not found - then create a
new connection -
            {
                conn = new ServerConnection(info);
                g_ServerConnections.addElement(conn);
            }

            conn.m_nRefCount++;          // increment the
reference count to this connection
            return conn;
        }

    /*****************************************************************
    ******************
         * Finds an existing ServerConnection that we could
    use for the requested connection.
         * Currently, we only use server, port and nick
    name.
         * We can either use the OriginalNick requested by
    the connection OR the nickname that
         * the connection is now using. NB: They can be
    different, if we ended up using an automatically
         * generated alternate name. Returns null if no
    suitable connection found
         */
        static private ServerConnection
    findServerConnection(ServerConnectionInfo info)
        {
            for (int i = 0; i < g_ServerConnections.size();
    i++)
            {
                ServerConnection conn = (ServerConnection)
    g_ServerConnections.elementAt(i);
                if
    (conn.getServer().equals(info.m_strServer)
                        && conn.getPort() == info.m_nPort
                        &&
    (conn.getNick().equals(info.m_strNick) ||
    conn.m_strOriginalNick.equals(info.m_strNick)))
                    return conn;
            }
            return null;
        }


/*****************************************************
*****************
        * Returns the key() that would look up this
element in the table of hash connections
        * Must match the lookup mechanism used above

******************************************************
****************/
    private String getKey()
    {
        return m_strOriginalNick + "@" + m_strServer;
    }

    private ServerConnection(ServerConnectionInfo info)
    {
        // TODO - move a ServerConnectionInfo instance
INTO this class
        m_strServer = info.m_strServer;
        m_nPort = info.m_nPort;
        m_strUser = info.m_strUser;
        m_strOriginalNick = info.m_strNick;
        m_strNick = m_strOriginalNick;
        m_strPass = info.m_strPass;
        m_strRealName = info.m_strRealName;
    }


    public String toString()
    {
            return super.toString() + "( host=" +
m_strServer + ":" + m_nPort
                  + " nick=" + m_strNick + " RefCount=" +
m_nRefCount + ")";
        }




/*****************************************************
****************
        * Debugging Function - list the connections and
the channels in use

*****************************************************
***************/
    static public void dumpInfo()
    {
        Enumeration enum =
g_ServerConnections.elements();
        while (enum.hasMoreElements())
        {
            ServerConnection conn =
(ServerConnection)enum.nextElement();
            log(conn.toString());
            for (int i = 0; i <
conn.m_vecChannels.size(); i++)
            {
                log ("\tchannel=" +
conn.m_vecChannels.elementAt(i));
            }
            for (int i = 0; i <
conn.m_vecListeners.size(); i++)
            {
                    log ("\tlistener=" +
    conn.m_vecListeners.elementAt(i));
                }
            }

        }

    /*****************************************************
    ****************
         * Close the Connection to the Server - handles
    sending a QUIT command.
         */
        public void close()
        {
            m_nRefCount--;                    // decrement the
    Ref count.
            if (m_nRefCount == 0)
            {
                postRaw("QUIT");                    // say we're
    quitting
                g_ServerConnections.removeElement(this);
    // find and remove this connection from the pool
                m_bStopped = true;            // flag to make
    threads terminate
            }
        }

    /*****************************************************
    ****************
         * starts the process of connecting to the server
```

```
*******************************************************
*****************/
        public void connect()
        {
            // start a server connection thread to begin
connecting to the server in
            // the backgroud
            if (m_threadSender == null)  // start the
sender thread - if not already started
            {
                m_threadSender = new
ServerConnectionThread();
                m_threadSender.setName("ChatSender");
                m_threadSender.start();
            }
        }


/*******************************************************
******************
        * Return the state of the connection. See STATE_
constants for details

*******************************************************
*****************/
        public int getConnectState()
        {
            return m_nConnectState;
        }
```

```
/*******************************************************
******************
      * Reset the count so we try to connect to the
server again

*******************************************************
*****************/
    private void forceConnect()
    {
        if (m_nConnectState == STATE_CONNECTION_FAILED)
        {
            m_nConnectState = STATE_NO_SOCKET;
            synchronized (m_queue)
            {
                m_queue.notify();             // to wake
up the sending thread, if it's waiting because of
connection failure
            }
        }
    }


/*******************************************************
******************
      * determines if the server is connected or not

*******************************************************
*****************/
    public boolean isConnected()
    {
        return false;
    }
```

```
        public String getNick()
        {
            return m_strNick;
        }

        public String getServer()
        {
            return m_strServer;
        }

        public int getPort()
        {
            return m_nPort;
        }

        public String getUser()
        {
            return m_strUser;
        }


    /*****************************************************
    *****************
         * places a raw message on the outgoing queue

    *****************************************************
    *****************/
        public void postRaw(String strRaw)
        {
            if (m_nConnectState == STATE_CONNECTION_FAILED)
            {
                forceConnect();      // another message
posted in externally - try to connect again
            }

            synchronized (m_queue)
            {
                System.err.println("Posting " + strRaw);
                m_queue.push(strRaw);
                m_queue.notify();
            }
        }

        private void startReceiverThread()
        {
            m_threadReceiver = new IncomingThread();
            m_threadReceiver.setName("ChatReceiver");
            m_threadReceiver.start();
        }


/*****************************************************
*****************
        * Processes incoming messages from the socket -
NB: this is technically happening
        * in a separate thread !

*****************************************************
***************/
        private void processMessage(String str)
        {
            System.err.println(str);          // for
debugging
            ChatMessage msg = ChatMessage.parseString(str);
            if (msg.getParameterCount() == 0)
                return;                // empty message -
ignore it.

            if (msg.getParam(0).equals("PING"))
            {   // if it's a PING, make an automatic
reponse...
                {
                    // respond to PING automatically...
                    try
                    {
                        m_threadSender.sendRaw("PONG " +
msg.getParam(1));
                    }
                    catch (IOException e)
                    {
                        e.printStackTrace();
                    }
                    return;
                }
            }

            // if it's a valid reply then that should
trigger we're active
            int nCode = msg.getResponseCode();
            if (m_nConnectState ==
STATE_AWAITING_LOGON_RESPONSE)
            {
                if (nCode != 0)
                {
                    if (nCode < Codes.ERROR_START)
                    {
                        m_nConnectState = STATE_ACTIVE;
                        }
                        else
                        {
                            if (m_nConnectErrorCode == 0)
                                m_nConnectErrorCode = nCode;
// set the reason for failure


                            // server returned some kind of
error - most likely a failure to logon
                            serverMessage("error connecting " +
msg.getLastParam());
                            // we don't need to close the
connection here. It should be done by the
                            // client as we're logging in.
                        }

                    // kick the logon semaphore - either
we've set the logon state OR we haven't (in which case
the
                    // ConnectErrorCode is set
                    synchronized(m_logonSemaphore)
                    {
                        m_logonSemaphore.notify();
                    }
                }
            }

            // inform anyone listening to this server...
            for (int i = 0; i < m_vecListeners.size(); i++)
            {
                ChatListener o =
(ChatListener)m_vecListeners.elementAt(i);
                if (o.chatEvent(msg))
```

```
                    break;
                }
            }

    /*******************************************************
****************
         * joins the specified channel, and returns the
    corresponding channel object
         * for the join - NB does NOT mean that the channel
    is a guaranteed mechansim for
         * communicating with the client.
         */
        public Channel joinChannel(String strChannel)
        {
            strChannel = "#" + strChannel;
            postRaw("JOIN " + strChannel);
            Channel channel = new Channel(this,
strChannel);
            m_vecChannels.addElement(channel);        // add
to the list of channels
            return channel;
        }

        void removeChannel(Channel channel)
        {
            m_vecChannels.removeElement(channel);
        }

        public void addListener(ChatListener listener)
        {
            m_vecListeners.addElement(listener);
        }
        public void removeListener(ChatListener listener)
        {
            m_vecListeners.removeElement(listener);
        }


        private void serverMessage(String strMsg)
        {
            serverMessage(0, strMsg);
        }

        private void serverMessage(int nCode, String
strMsg)
        {
            // tell the listeners....
            for (int i = 0; i < m_vecListeners.size(); i++)
            {
                ChatListener o =
(ChatListener)m_vecListeners.elementAt(i);
                o.serverMessage(nCode, strMsg);
            }
        }


    /***************************************************
****************
        * Very simple thread to read from the incoming
socket stream. Just passes the results
        * through to the callback in the parent.
        */
    private class IncomingThread extends Thread
    {
            public void run()
            {
                System.out.println("IncomingThread
    starting");

                while (!m_bStopped)
                {
                    try
                    {
                        String str = m_receive.readLine();
                        // getting NULL normally means the
    socket is finished
                        if (str == null)
                            throw new
    IOException("readLine() returned null");

                        try
                        {
                            processMessage(str);
                        }
                        catch (Throwable e)
                        {

    //ChatInstance.reportException(e);  // an exception
    here should NOT kill us - report it, because it needs
    investigating !
                        }
                    }
                    catch (InterruptedIOException e)
                    {
                        // silently swallow this message -
    done so we check the stopped flag
                        // every so often
                        }
                        catch (Exception e)
                        {
                            // can get an exception
legitimately, if the stopped flag is set.
                            // Tempting to just try reading
from the socket again, but this might
                            // create an infinite loop, so we
force the socket closed and force
                            // a re-logon to the chat server at
the next attempt.
                            if (!m_bStopped)
                            {
                                System.err.println("Exception
in SocketListener thread: " + e);
                                e.printStackTrace();
                                m_threadSender.close();
// close the connection in the sending thread
                                // we should also fake a
connection to the interface by means of a servermessage
                                serverMessage("No connection to
server");
                            }
                            break;
                        }
                    }
                    System.out.println("IncomingThread ended");
                }
            }


//**************************************************************
******************
```

```
        private class ServerConnectionThread extends Thread
        {

            public final int CONNECTION_TRY_LIMIT = 3;
    // no of successive attempts to connect to specified
    server
            public final int ALTERNATE_NICK_TRY_LIMIT = 5;
    // no of auto-generated alternate Nicknames we try.
            private Socket m_sock;
            private OutputStreamWriter m_send = null;
    // out-going stream

            public void run()
            {
                // first job is to try and connect.
                connect();

                String strMsg = null;
                while (true)
                {
                    synchronized (m_queue)
                    {
                        try
                        {
                            // if the the queue is empty,
    OR we failed to connect, then keep waiting.
                            while (!m_bStopped &&
    (m_queue.isEmpty() || m_nConnectState ==
    STATE_CONNECTION_FAILED))
                                m_queue.wait();
                        }
                        catch (InterruptedException e)
                        {
                            break;
                        }
                        // the test below means that
provided we have an active connection,
                        // we will send ALL the messages in
the queue, before we exit.
                        // E.g when closing the window, we
need to send a "PART", etc.
                        if (m_bStopped && (
m_queue.isEmpty() || m_nConnectState != STATE_ACTIVE))
                            break;

                        strMsg = (String)m_queue.pop();
                    }
                    if (!sendRobust(strMsg))
                    {
                        // this method is synchronized - so
no need to sync around it.
                        m_queue.pushHead(strMsg);    //
failed to send the message - push it back on the queue
                    }
                }
                close();                  // close the sockets,
etc
                System.out.println("Server Connection
Thread terminating");
            }


/*****************************************************
***************
```

```
            * Attempts to connect repeatedly until we're
done
            * @return true if connection successful
***********************************************************
*************/
        private boolean connect()
        {
System.out.println("ServerConnection.connect()
nConnectState=" + m_nConnectState);
            if (m_nConnectState == STATE_NO_SOCKET)
            {
                int nTries = 0;
                int nAlternateNick = 0;
                while (nTries < CONNECTION_TRY_LIMIT)
                {
                    if (connectOnce())
                        return true;        // done -
we are now connected

                    // if we got a nickname collision -
try an alternative...
                    // if we got a legit error code
from the server - there is no point blindly retrying.
                    // if it was a nickname collision,
we need to try a new nickname
                    if (m_nConnectErrorCode != 0)
                    {
                        if (m_nConnectErrorCode ==
Codes.ERR_NICKCOLLISION || m_nConnectErrorCode ==
Codes.ERR_NICKNAMEINUSE)
                        {
```

```
                                        if (nAlternateNick <
ALTERNATE_NICK_TRY_LIMIT)
                                        {
                                            nAlternateNick++;
                                            m_strNick =
m_strOriginalNick + "_" +
Integer.toString(nAlternateNick);
                                            continue;       //
doesn't count as a failure to retry.
                                        }
                                    }
                                }
                                nTries++;
                            }
                            m_nConnectState =
STATE_CONNECTION_FAILED;
                            serverMessage("repeatedly unable to
connect...giving up");
                            return false;
                        }
                        else if (m_nConnectState ==
STATE_CONNECTION_FAILED)
                            return false;       // failed before -
still failed now
                        else
                            return true;
                }


/*****************************************************
**************
```

```
             * Sends the message robustly. If we get an
    IOException, we automatically
             * try to reconnect to the server and re-send
    the message
             */
            private boolean sendRobust(String strMsg)
            {
                System.err.print("sendRobust(" + strMsg +
    ")");

                int nTries = 0;
                while (true)
                {
                    if (!connect())             // we
    failed to connect. Returning false, means the message
    wasn't sent
                        return false;

                    try
                    {
                        sendRaw(strMsg);
                        return true;      // OK
                    }
                    catch (IOException e)
                    {
                        // some kind of error writing to
    the socket - assume it's closed
                        // and we need to open another - so
    shut down and re-open it
                        close();

                        // otherwise try to connect again
                        nTries++;
                    if (nTries > CONNECTION_TRY_LIMIT)
                    {
                        m_nConnectState =
STATE_CONNECTION_FAILED;
                        serverMessage("not connected");
                        return false;
                    }
                }
            }
        }


        // helper method - only intended so we send all
messages to the underlying
        // stream from one place (easy to add logging,
etc)
        final private void sendRaw(String strMsg)
throws IOException
        {

System.out.println("ServerConnection.sendRaw(" + strMsg
+ ")");
            //m_send.println(strMsg);
            m_send.write(strMsg);
            m_send.write('\n');     // each line must
end in newline
            m_send.flush();
        }


        public boolean connectOnce()
        {
            serverMessage("Connecting to " +
m_strServer + ":" + m_nPort);
            m_nConnectState = STATE_NO_SOCKET;
            // create the socket and listen on it.
            try
            {
                m_sock = new Socket(m_strServer,
m_nPort);
                m_receive = new BufferedReader(new
InputStreamReader(m_sock.getInputStream()));
                m_send = new
OutputStreamWriter(m_sock.getOutputStream());
//              m_send = new
PrintWriter(m_sock.getOutputStream(), true);
                m_nConnectState =
STATE_AWAITING_LOGON_RESPONSE;
                m_nConnectErrorCode = 0;

                // send in opening messages - PASS,
NICK, USER
                m_strHost =
InetAddress.getLocalHost().toString();

                if (m_strPass != null &&
m_strPass.length() > 0)
                    sendRaw("PASS " + m_strPass);

                if (m_strNick != null &&
m_strNick.length() > 0)
                    sendRaw("NICK " + m_strNick);
                    // if no full name, then use the
nickname
                    sendRaw("USER " + m_strUser + " " +
m_strHost
                                   + " " + m_strServer + "
:" + m_strRealName);

                    // lastly send an identification of our
software version no - this is an extension to IRC.
                    sendRaw("CLIENTVERSION " +
ChatMessage.PROTOCOL_VERSION + " " + getCVSVersion());
                }
                catch (IOException e)
                {
                    serverMessage("Connection failed " +
e.getMessage());
                    System.out.println(e.getMessage());
                    e.printStackTrace();
                    close();
                    return false;
                }


            // fire off the thread to read from this
socket
            startReceiverThread();

            serverMessage("registering on server");
            // now we want to wait until logged on,
before we attempt to do anything else
            // the login state will be set by the
receiver thread. We use m_logonSemaphore
```

```
                // to synchronize the results. We only give
the receiver thread 10 seconds to
                // get a reply, before regarding the logon
as unsuccessful. This isn't very long
                synchronized (m_logonSemaphore)
                {
                    if (m_nConnectState ==
STATE_AWAITING_LOGON_RESPONSE)
                    {
                        try
                        {
                            m_logonSemaphore.wait(20 *
1000);              // wait until we are notified on this
object
                        }
                        catch (InterruptedException e)
                        {

System.out.println("InteruptedException on connect()" +
e);
                        }
                    }
                }
                if (m_nConnectState != STATE_ACTIVE)     //
if not active now, close down and go away
                {
                    close();
                    serverMessage("connection failure on
logon - no response from server");
                    return false;
                }

                // also JOIN all the channels we had before
                // NB: Possible collision error here, as we
could be in the process of JOINING
                // one of these channels on the servser

                try
                {
                    for (int i = 0; i <
m_vecChannels.size(); i++)
                    {
                        Channel channel =
(Channel)m_vecChannels.elementAt(i);
                        sendRaw("JOIN " +
channel.getName());
                    }
                }
                catch (IOException e)
                {
                    close();
                    return false;           // error in
sending the RAW messages
                    // didn't want to risk recursion by
calling sendRobust() here !
                }
                return true;
            }



/*****************************************************
*************
        * Make sure the streams and the socket is
closed
```

```
         */
        private void close()
        {
            if (m_receive != null)
            {
                try
                {
                    m_receive.close();
                }
                catch (Exception e)
                {
System.err.println("ServerConnection.close() - error
closing input stream " + e);
                }
                m_receive = null;
            }

            if (m_send != null)
            {
                try
                {
                    m_send.close();
                }
                catch (Exception e)
                {
System.err.println("ServerConnection.close() - error
closing output stream " + e);
                }
                m_send = null;
            }
```

```
                    if (m_sock != null)
                    {
                        try
                        {
                            m_sock.close();
                        }
                        catch (Exception e)
                        {

System.err.println("ServerConnection.close() - error
closing socket " + e);
                        }
                        m_sock = null;
                    }
                    m_nConnectState = STATE_NO_SOCKET;
// no socket
                }
            }


        static private void log(String str)
        {
            System.out.println(str);
        }

    }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatclient/ServerConnec
tionInfo.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatclient;

/**
 * Abstracts information required for server connection
- should probably have getter/setter, etc
 * $Revision: 1.3 $
 * $Date: 2002/04/15 22:25:12 $
 * $Author: markc $
 */

public class ServerConnectionInfo implements Cloneable
{
    public String m_strServer = "";
    public int    m_nPort = 6667;
    public String m_strUser = "";
    public String m_strNick = "";
    public String m_strPass = "";
    public String m_strRealName = "";        // the real
name

    public ServerConnectionInfo()
    {
    }

    public boolean equals(Object obj)
    {
        if (obj instanceof ServerConnectionInfo)
        {
                ServerConnectionInfo s =
    (ServerConnectionInfo)obj;
                return (m_strServer.equals(s.m_strServer)
                    && m_nPort == s.m_nPort
                    && m_strUser.equals(s.m_strUser)
                    && m_strNick.equals(s.m_strNick)
                    && m_strPass.equals(s.m_strPass));
            }
            else
                return false;
        }

        public String toString()
        {
            return ("server=" + m_strServer + " port=" +
    m_nPort
                    + " user=" + m_strUser + " nick=" +
    m_strNick
                    + " pass=" + m_strPass + " realname=" +
    m_strRealName);
        }


    /*******************************************************
    ******************
        * More friendly version of clone()
        */
       public Object clone()
       {
           try
           {
               return super.clone();
           }
```

```
                catch (CloneNotSupportedException e)
                {
                    throw new Error("ClonneNotSupported");
        // can't happen
                }
            }
        }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatclient/Test.java
//++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatclient;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */

public class Test
{
    public Test()
    {
    }

    public static void main(String[] args)
    {
        ServerConnection server =
ServerConnection.getServerConnection( "localhost",
6667, "user", "nick", "pass");
        server.connect();
        Channel channel = server.joinChannel("talk");
```

```
            ChatPane chatPane = new ChatPane();
            chatPane.setChannel(channel);

            // create a frame for it
            JFrame frame = new JFrame("ChatPane");

            frame.addWindowListener(new WindowAdapter()
            {
                public void windowClosing(WindowEvent e)
{System.exit(0);}
            });

frame.getContentPane().add(chatPane.getPanel());
            frame.pack();
            frame.setVisible(true);


            try
            {
                Thread.sleep(10000);
            }
            catch (InterruptedException e)
            {
            }
        }
    }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/ActionDel
egator.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

/**
 * Provides an Action implementation that wraps an
existing Action implementation.
 * It delegates to the wrapped action for the
actionPerformed().
 *
 * For getValue() it retrieves the wrapped actions
value, if no value is specifically
 * set for this object (using putValue()).
 *
 * The reason for this class is that the standard
JTextComponent actions are shared
 * and changing their properties has a global effect,
which might impact other software.
 * Basically, all we wanted to change was their names
which appear on menus, and also
 * to have some confidence that they wouldn't be
impacted by setEnabled().
 *
 * @author $Author: markc $
 * @version $Revision: 1.1 $
 * $Date: 2002/03/19 17:23:45 $
 */

import java.beans.PropertyChangeListener;
import java.awt.event.ActionEvent;
```

```
    import javax.swing.AbstractAction;
    import javax.swing.Action;


    public class ActionDelegator extends AbstractAction
    {
        private Action m_action;

        public ActionDelegator(Action action)
        {
            m_action = action;
        }

        public Object getValue(String key)
        {
            // if set explicitly in this object, return it
  - otherwise check to
            // see if the parent had a value.
            Object obj = super.getValue(key);
            if (obj != null)
                return obj;
            else
                return m_action.getValue(key);
        }

        public void actionPerformed(ActionEvent e)
        {
            m_action.actionPerformed(e);
        }
    }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/AskExecut
eVerbolPanel.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

import javax.swing.*;
import java.awt.*;

/**
 * Simple Panel for showing in a JOptionPane to ask the
user whether the verbol
 * just received should be executed or not. Includes a
checkbox to let them
 * specify if input from the same user should be
accepted next time
 * $Id: AskExecuteVerbolPanel.java,v 1.1 2002/02/21
21:25:09 markc Exp $
 */

public class AskExecuteVerbolPanel extends JPanel
{
    JCheckBox m_checkAccept;

    public AskExecuteVerbolPanel(String strFrom, String
strVerbol)
    {
        this.setLayout(new BoxLayout(this,
BoxLayout.Y_AXIS));

        // has to be done as separate labels to get the
formatting - not very nice.
```

```
            this.add(new JLabel(strFrom + " has sent the
    verbol command"));

            JLabel label = new JLabel(strVerbol);

    label.setBorder(BorderFactory.createEmptyBorder(20, 20,
    20, 20));          // to indent and offset the verbol
            this.add(label);


            this.add(new JLabel("Would you like to execute
    this ?"));
            this.add(Box.createRigidArea(new Dimension(0,
    5)));
            m_checkAccept = new JCheckBox("Accept Verbol
    from " + strFrom + " without asking next time");
            this.add(m_checkAccept);
            //pack();
        }


    /*****************************************************
    *******************
         * Return true if they should accept next time
    without asking

    *****************************************************
    ******************/
        public boolean acceptNextTime()
        {
            return m_checkAccept.isSelected();
        }
```

```
}
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/Backgroun
dLoader.java
//+++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

/**
 * Thread for background loading the Swing classes. An
attempt to reduce the
 * startup time for loading swing.
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */

public class BackgroundLoader extends Thread
{
    private String[]  arrClasses = {
"javax.swing.JFrame",

"java.awt.TextField",

"javax.swing.JTable",
                                              };


    public BackgroundLoader()
    {
    }
/*******************************************************
*****************
         * Just load the classes and then stop
         */
        public void run()
        {
            for (int i = 0; i< arrClasses.length; i++)
            {
                String strClass = arrClasses[i];
                try
                {
                    Class.forName(strClass);
                    System.out.println ("loaded " +
strClass);
                }
                catch (ClassNotFoundException e)
                {
                    System.err.println("Class not found: "
+ strClass);
                }
            }
        }


    }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/ChatIconF
rame.java
//+++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import ChatInstance;
import java.net.URL;

/**
 * Frame displaying a grid of ChatIcons (Emoticons)
 * @author $Author: markc $
 * @version $Revision: 1.3 $
 * $Date: 2002/04/15 08:53:01 $
 */

public class ChatIconFrame extends JDialog implements
ActionListener
{
    private ChatInstance m_chatinstance;    // the
window this one sends events to !


    public ChatIconFrame(ChatInstance instance)
    {
        super(instance.getFrame(), "Chat Icons");
        m_chatinstance = instance;
        constructControls();
```

```
            pack();
            setVisible(true);
        }

        private void constructControls()
        {
            Container content = this.getContentPane();

            // use a gridlayout - 4 columns for the buttons
            int nCols = 4;
            content.setLayout(new GridLayout(0 , nCols));



            // we use the icon map to build up a list of
buttons. We want to only
            // display 1 button for each image, even though
there might be multiple
            // strings that generate that image. Also, we
want some control over the
            // order they are displayed. Used to use a
Hashtable, but that actually
            // gave different button orders on different
JVM's. Only disadvantage is
            // that the icon map has to have the icons in
file order.

            String[][] iconmap = ChatTextPane.getIconMap();
            int i = 0;
            while (i < iconmap.length)
            {
                String strImage = iconmap[i][1];
                String strFirstAbbrev = iconmap[i][0];
                String strAbbrev = iconmap[i][0];
                i++;
                // while the image is the same...
                while (i < iconmap.length &&
iconmap[i][1].equals(strImage))
                {
                    strAbbrev += " " + iconmap[i][0];
                    i++;
                }

                // now have a list of the abbrev and the
first abbrev - so can create the button
                URL url = getClass().getResource("/images/"
+ strImage);
                if (url == null)
                {
                    System.out.println ("** ERROR couldn't
load " + strImage);
                }
                else
                {
                    JButton button = new JButton(new
ImageIcon(url));
                    button.setMargin(new Insets(2,2,2,2));
                    button.setToolTipText(strAbbrev);

button.setActionCommand(strFirstAbbrev);
                    button.addActionListener(this);
                    content.add(button);
                }
            }

        }
        public void actionPerformed(ActionEvent e)
        {
            // when the user clicks on a button, we need to
    send the text through to
            // the entry field
            Object source = e.getSource();
            if (source instanceof JButton)
            {
                JButton btn = (JButton)source;
                String strText = " " +
    e.getActionCommand();

    m_chatinstance.insertEntryFieldText(strText);
            }
        }


    }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/ChatModel
.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

/**
 * Implements a chat model.
 * Keeps track of users, messages, etc.
 * @author
 * @version $Revision: 1.19 $ $Date: 2002/04/24
16:35:59 $
 */
import java.util.*;
import com.fs.chatclient.*;
import javax.swing.table.AbstractTableModel;
import javax.swing.AbstractListModel;
import javax.swing.table.TableModel;
import javax.swing.event.ListDataListener;
import javax.swing.ListModel;
import javax.swing.DefaultListModel;
import java.text.SimpleDateFormat;
import ChatExtension;

public class ChatModel implements ChatListener
{
    static public final int MSGTYPE_INCOMING = 1;
// comes to this user
    static public final int MSGTYPE_SERVERNICE = 2;
// from the server (e.g. "xxx has joined");
    static public final int MSGTYPE_SERVERREPLY = 3;
// numeric reply code from the server
```

```
        static public final int MSGTYPE_INTERNAL = 4;
    // internal within the software (e.g. connecting to
    ....)
        static public final int MSGTYPE_OUTGOING = 5;
    // sent from this user



        private UserTableModel m_modelUsers = new
    UserTableModel(); // model for the users table

        public String m_strNickToFollow = "";
    // nick we should be following
        private String  m_strTopic = "";
    // current topic for this char room

        private Channel m_channel = null;
    // channel to use

        private ChatModelVerbolListener m_ui;
    // ui to call back into for certain events

        private MessageListModel m_msgList = new
    MessageListModel();

        static private ChatModel g_NullModel = new
    ChatModel();          // the one and only NULL chat model


    /*****************************************************
    ********************
        * Create a model for a channel on a specific
    server
```

```
****************************************************************
******************/
    static public ChatModel
joinChannel(ServerConnection conn, String strChannel)
    {
        ChatModel model = new ChatModel();

        conn.addListener(model);
        model.m_channel = conn.joinChannel(strChannel);

        return model;
    }


   // gets a model that chats to nothing !
   static public ChatModel getNullChannel()
   {
       return g_NullModel;
   }


/****************************************************************
*******************
    * Returns true if this is THE null model
    */
   public boolean isNull()
   {
       return (this == g_NullModel);
   }

   private ChatModel()
   {
        }


        public String toString()
        {
            return ("channel = " +
    getChannel().toString());
        }


/***********************************************************************
******************
        * Returns the underlying Channel connection for
this channel
        */
        public Channel getChannel()
        {
            return m_channel;
        }


/***********************************************************************
******************
        * Returns the topic currently set for this channel

************************************************************************
*****************/
        public String getTopic()
        {
            return m_strTopic;
        }

        public UserTableModel getUsersTableModel()
        {
            return m_modelUsers;
        }



        public MessageListModel getMessageListModel()
        {
            return m_msgList;
        }

        public void
    setVerbolListener(ChatModelVerbolListener listener)
        {
            m_ui = listener;
        }


    /*******************************************************
    ******************
        * implement the ChatListener interface
        */

        public void serverMessage(int nCode, String strMsg)
        {
            appendMessage("", strMsg,
    ChatModel.MSGTYPE_INTERNAL);
        }



    /*******************************************************
    ******************
```

```
        * Handle an incoming message from the chat server
    connection
        */
       public boolean chatEvent(ChatMessage msg)
       {
           // NB: since this routine won't be called if
    m_channel is NOT NULL, then
           // we don't need to ensure it must work.
           if (m_channel == null)
           {
               System.err.println("chatEvent called with
    null channel");
               return false;
           }

           int nCode = msg.getResponseCode();
           if (nCode != 0)
           {
               return handleChatCodes(nCode, msg);
           }

           String strCommand = msg.getParam(0);
           String strChannel = msg.getParam(1);
           if (strCommand.equals("PRIVMSG") ||
    strCommand.equals("NOTICE"))
           {
               if (m_channel.getName().equals(strChannel)
                   || getMyNick().equals(strChannel))
               {   // message for this channel or
    explicitly to this user
                   msg.parseTime();                // look
    to extract the time (if not already done so).
                   String strMessage = msg.getLastParam();
```

```
                    strMessage = strMessage.trim();     //
remove leading/trialing whitespace

                    String strFrom = getFromAndTime(msg);
                    if (getMyNick().equals(strChannel))
                        strFrom += "[PRIVATE]";     //
indicate this was sent just to this Nick - not to the
channel
                    if (isVerbolMessage(strMessage))
                    {
                        if
(ChatExtension.getInstance().m_bShowVerbol)
                        {
                            appendMessage(strFrom,
strMessage, MSGTYPE_INCOMING);
                        }
                        handleVerbol(msg);
                    }
                    else
                        appendMessage(strFrom, strMessage,
MSGTYPE_INCOMING);

                    return true;
                }

            }
            else if (strCommand.equals("JOIN"))
            {
                if (m_channel.getName().equals(strChannel))
                {
                    // someone else has joined
                    appendMessage("", msg.getContributor()
+ " has joined", MSGTYPE_SERVERNICE);
```

```
m_modelUsers.addUser(msg.getContributor());
                        requestRealName(msg.getContributor());
                        return true;
                  }
            }
            else if (strCommand.equals("PART"))
            {
                  if (m_channel.getName().equals(strChannel))
                  {
                        // someone has left
                        // Some servers might display a reason
for the user having left. We attempt to include
                        // this.
                        String strReason = "";
                        if (msg.getParameterCount() > 2)
                        {
                              strReason = msg.getLastParam();
                        }
                        appendMessage("", msg.getContributor()
+ " has left " + strReason, MSGTYPE_SERVERNICE );

m_modelUsers.removeUser(msg.getContributor());
                        return true;
                  }
            }

            return false;
      }


/*****************************************************
*******************
```

```
        * Helper function - handles chat response codes.
    Only called from chatEvent()
        * above
        */
     private final boolean handleChatCodes(int nCode,
    ChatMessage msg)
     {
         switch (nCode)
         {
             case Codes.RPL_NAMREPLY:
             {
                 // to get dIRC working the RPL_NAMRPL
    includes an equals before the name of the
                 // channel. This doesn't seem to be in
    the spec, but we ensure we can handle it
                 // here
                 String strChannel = msg.getParam(2);
                 if (strChannel.equals("="))
                     strChannel = msg.getParam(3);

                 if
    (m_channel.getName().equals(strChannel)) // make sure
    it is to this channel
                 {
                     // build a vector containing the
    names now in the group.

                     // when reading the names list, we
    need to make sure we copy across
                     // information we know about
    existing users.
                     StringTokenizer tok = new
    StringTokenizer(msg.getLastParam(), " ");
```

```
                        Vector vecUsers = new Vector();
                        while (tok.hasMoreElements())
                        {
                            String strName =
 tok.nextToken();
                            strName =
 stripOperatorEtcFromName(strName);
                            vecUsers.addElement(strName);
                        }

                        // then update the model
                        String newUsers =
 m_modelUsers.newUserList(vecUsers);
                        if (newUsers.length() > 0)
                        {
                            requestRealName(newUsers);
                        }

                        return true;
                    }
                    break;
                }

                case Codes.RPL_WHOISUSER:
                {
                    //311      RPL_WHOISUSER
                    //         "<nick> <user> <host> *
 :<real name>"
                    String nick = msg.getParam(2);
                    String realname =
 msg.getFreeFormParam();
                    if (nick.length() > 0 && realname !=
 null && realname.length() > 0)
```

```
                    {
                        m_modelUsers.setUserRealName(nick,
realname);
                    }
                    break;
                }

                // set the topic - probably not a lot of
call for this !
                case Codes.RPL_TOPIC:
                {
                    if
(m_channel.getName().equals(msg.getParam(2)))
                    {
                        m_strTopic = msg.getLastParam();
                        return true;
                    }
                }

                case Codes.RPL_ENDOFMOTD:              // hide
some of the more inane messages
                case Codes.RPL_ENDOFUSERS:
                    break;

                default:
                {
                    appendMessage("", msg.getLastParam(),
MSGTYPE_SERVERREPLY);  // default is to display the
last parameter

                }
            }
            return false;
```

```
        }

    /***************************************************************************
         * sends a WHOIS command to the server to request
    information on the specified NICK.
         * @param names a comma-separated list of nicknames
    to find out about
         */
        void requestRealName(String names)
        {
            m_channel.getServerConnection().postRaw("WHOIS
    " + names);
        }

        static boolean isVerbolMessage(String strMsg)
        {
            return strMsg.startsWith("VERBOL!");
        }

    /************************************************************************
         * handles the case of a ChatMessage that starts
    with the VERBOL! string
         */
        private void handleVerbol(ChatMessage msg)
        {
            // read the tokens, but skip the initial
    "VERBOL!" string
```

```
            String strVerbol =
    msg.getLastParam().substring(7);
            StringTokenizer tok = new
    StringTokenizer(strVerbol, " ");

            if (!tok.hasMoreTokens())
                return;     // empty Verbol string

            String strVerb = tok.nextToken().toUpperCase();
            if (ChatVerbol.isSpecialVerb(strVerb))
            {
                if (tok.hasMoreElements())
                {
                    String strLoc = tok.nextToken();
                    // set the location of this contributor
    to be this location

    m_modelUsers.setUserActionLocation(msg.getContributor()
    , strVerb.toLowerCase(), strLoc);

                    // if it's a BROWSES and it matches the
    person we're following



                    if
    (m_strNickToFollow.equals(msg.getContributor())
                                &&
    (strVerb.equals(ChatVerbol.BROWSES_VERB)
                                    ||
    strVerb.equals(ChatVerbol.REVISED_VERB)
                                    ||
    strVerb.equals(ChatVerbol.COMPLETEDREVISION_VERB)))
```

```
                        {
                            boolean bRefresh =
    strVerb.equals(ChatVerbol.REVISED_VERB)
                                            ||
    strVerb.equals(ChatVerbol.COMPLETEDREVISION_VERB);

                            if (m_ui != null)
                                m_ui.navigateTo(strLoc,
    bRefresh);
                        }
                        // TODO for REVISION, need to ask the
    ui to refresh the current page
                    }
                    return;          // "BROWSES" is not real
    verbol so don't ask about executing it
                }
                else
                {
                    // ask ui to execute real verbol
                    if (m_ui != null)

    m_ui.executeVerbol(msg.getContributor(), strVerbol,
    false);
                }
            }


    /*****************************************************
    *****************
        * Strip leading "@", "+" from the user name
        */
        static private String
    stripOperatorEtcFromName(String strName)
        {
            for (int i = 0; i < strName.length(); i++)
            {
                if (strName.charAt(i) != '@' &&
    strName.charAt(i) != '+')
                    return strName.substring(i);
            }
            return "?";          // unknown user name
        }



    /*****************************************************
    ******************
        * close the model - remove connection to the
    channel/server, etc.
        */
        public void close()
        {
            if (m_channel != null)
            {
                m_channel.closeChannel(null);

    m_channel.getServerConnection().removeListener(this);
    // remove us from those interested in the connection
                m_channel.getServerConnection().close();
    // close the server connection
            }
        }
```

```
/*****************************************************
*****************
        * convenience method, obtains the nickname this
person is currently using
        */
    public final String getMyNick()
    {
        if (m_channel == null)
            return ("");
        else
            return
m_channel.getServerConnection().getNick();
    }


/*****************************************************
****************
        * Sets the location of this specified user. Called
when the user browses
        * to a new location, and when we obtain their
initial location
        */
    public void setMyLocation(String strUrl)
    {
        // explicitly set the action to browsers if it
wasn't supplied
        // m_modelUsers.setUserLocation(getMyNick(),
strUrl);
        m_modelUsers.setUserActionLocation(getMyNick(),
"browses", strUrl);
    }
```

```
/*****************************************************
***************
       * Sends an outgoing verbol navigation message to
the chat room. Includes
       * sending an outgoing message and updating the
chat room location. Called when
       * responding to an internal event from the ibook
controller that indicates a
       * change of location, etc.
       * @param verb Verb e.g. "BROWSES",
"COMPLETEDREVISION", etc.
       * @param url the location to browse/edit.

*****************************************************
***************/
    public void sendVerbolNavigationMessage(String
verb, String url)
    {
        m_modelUsers.setUserActionLocation(getMyNick(),
verb.toLowerCase(), url);

sendMessageToChannel(ChatVerbol.VERBOL_INDICATOR + " "
+ verb + " " + url);
    }




/*****************************************************
***************
     * sends an outbound message. needs to both post the
message to the server
```

```
        * and add it into the model of messages to be
    displayed to the user
        */
        public void sendOutgoingMessage(String strTo,
    String strText)
        {
            if (m_channel == null)
                return;

            if (strText.length() == 0)
                return;         // no message - do nothing.

            if (strText.charAt(0) == '/' &&
    strText.length() > 1)
                // if it begins with SLASH - treat rest of
    text as a raw server command (strTo is irrelevant)
            {

    m_channel.getServerConnection().postRaw(strText.substri
    ng(1));
                appendMessage(getMyNick(), strText,
    MSGTYPE_OUTGOING);
            }
            else
            {
                // if hiding verbol commands - then that
    also includes any OUTGOING commands from us.
                // TODO - should only hide the implicit
    bits of Verbol e.g. BROWSES, REVISES, REVISED
                if
    (ChatExtension.getInstance().m_bShowVerbol ||
    !isVerbolMessage(strText))
                {
                    if (strTo.charAt(0) == '#') // it's
going to a group
                        appendMessage(getMyNick(), strText,
MSGTYPE_OUTGOING);
                    else
                        appendMessage(getMyNick() +
"[PRIVATE TO " + strTo + "]", strText,
MSGTYPE_OUTGOING);
                }
                String strMsg = "PRIVMSG " + strTo + " :" +
strText;

m_channel.getServerConnection().postRaw(strMsg);
            }
        }


/*****************************************************
****************
     * Sends the specified message to the Channel.
     */
    public void sendMessageToChannel(String strText)
    {
        if (m_channel != null)
        {
            sendOutgoingMessage(m_channel.getName(),
strText);
        }
    }

    private void appendMessage(String strFrom, String
strMsg, int nMsgType)
    {
            // add to our model
            m_msgList.addMsg(strFrom, strMsg);
            // forward to the UI to handle aswell (e.g. add
    to text field, etc)
            if (m_ui != null)
                m_ui.textMessage(strFrom, strMsg,
    nMsgType);
        }



    /****************************************************
    ****************
        * Requests a list of names for this channel.
    Handling the RPL_NAMREPLY response
        * will update the list of users asynchronously
        */
        public void refreshUsers()
        {
            if (m_channel != null)
            {
                String strMsg = "NAMES " +
    m_channel.getName();

    m_channel.getServerConnection().postRaw(strMsg);
            }
        }



        static public class UserInfo
        {
            private String m_strNick;
            public String m_strUrl;
            public boolean m_bOperator;
            public String m_strAction;      // last known
    verbol action
            public String m_strRealName;    // real name

            UserInfo(String strNick)
            {
                m_strNick = strNick;
            }

            public String getNick()
            {
                return m_strNick;
            }


    /*************************************************************
    ****************
             * set the real name for this user
             */
            public void setRealName(String name)
            {
                m_strRealName = name;
            }


    /*************************************************************
    ****************
             * return the display name for the user. This
    consists of the nickname,
         * plus the real name in brackets
         */
        public String getDisplayName()
        {
            String str = m_strNick;
            if (m_strRealName != null)
                str += " (" + m_strRealName + ")";
            return str;
        }


/*********************************************************
****************
         * Returns the "real" name if there is one
         */
        public String getRealName()
        {
            return m_strRealName;
        }


        public String toString()
        {
            return m_strNick;
        }

        static public class CompareByNick implements
QuicksortAlgorithm.Comparator
        {
            public int compare(Object o1, Object o2)
            {
                // JDK 1.1 doesn't even have
compareToIgnoreCase() !!!
```

```
                    return
((UserInfo)o1).m_strNick.toLowerCase().compareTo(((User
Info)o2).m_strNick.toLowerCase());
               }
          }


     }


     //
*******************************************************
***************
     public class UserTableModel extends
AbstractTableModel
     {
          private Hashtable m_hashUsers = new
Hashtable();          // hashtable of UserInfo
          private Vector m_vecKeys = new Vector();
// vector of UserInfo
          private String[] m_strColumnName = { "user",
"action", "location" };

          public UserTableModel()
          {
          }


/*******************************************************
***************
          * call this when changing the hash table in
the outer class, so we can
```

```
         * construct the keys. Need to do this to
provide a mapping of the keys
         */
        public void update()
        {
            UserInfo[] arrUsers = new
UserInfo[m_hashUsers.size()];
            Enumeration enum = m_hashUsers.elements();
            int i = 0;
            while (enum.hasMoreElements())
            {
                arrUsers[i] =
(UserInfo)enum.nextElement();
                i++;
            }
            QuicksortAlgorithm.sort(arrUsers, new
UserInfo.CompareByNick());

            // now copy into the vector of keys
            m_vecKeys = new Vector(arrUsers.length);
            for (i = 0; i < arrUsers.length; i++)
            {
                m_vecKeys.addElement(arrUsers[i]);
            }
            fireTableDataChanged();
        }


        public int getRowCount()
        {
            return m_hashUsers.size();
        }
        public int getColumnCount()
        {
            return 3;    // always 2
        }

        public String getColumnName(int columnIndex)
        {
            return m_strColumnName[columnIndex];
        }

        /*
        public Class getColumnClass(int columnIndex)
        {
            return new String().getClass();
        }*/


/*******************************************************
***************
        * This is the model override used to display
the data in the table.
        * Slight abuse of MVC as we modify the text
returned
        */
      public Object getValueAt(int rowIndex, int
columnIndex)
      {
          UserInfo u =
(UserInfo)m_vecKeys.elementAt(rowIndex);
          switch (columnIndex)
          {
              case 0:
                    return u.getDisplayName();    // NB:
    used by the rendered.

                case 1:
                    return u.m_strAction;

                case 2:
                    return u.m_strUrl;
            }
            return null;            // shouldn't happen
        }

        public String getNickAt(int nIndex)
        {
            return
    ((UserInfo)m_vecKeys.elementAt(nIndex)).m_strNick;
        }

        public String getUserLocation(String strNick)
        {
            return ((UserInfo)
    m_hashUsers.get(strNick)).m_strUrl;
        }


        public String getNickToFollow()
        {
            return m_strNickToFollow;
        }


/*******************************************************
*************
```

```
              * Adds a user to the model
              */
             UserInfo addUser(String str)
             {
                 UserInfo u =
        (UserInfo)m_hashUsers.get(str);
                 if (u == null)
                 {
                     u = new UserInfo(str);
                     m_hashUsers.put(str, u);
                     update();
                 }
                 return u;
             }


      /***********************************************************
      ************
              * Removes the specified user from the model
              */
             void removeUser(String strUser)
             {
                 m_hashUsers.remove(strUser);
                 update();
             }


      /***********************************************************
      ************
              * Updates the hashtable to have the users
      specified in the hash table,
              * retaining details (ie locations) of all the
      users in the existing list
```

```
         * @param vecUsers Strings of user names
         * @return String containing comma-separated
list of NEW names - used to construct WHOIS command
         */
        String newUserList(Vector vecUsers)
        {
            StringBuffer buf = new StringBuffer();
            Hashtable hashNewUsers = new Hashtable();
            for (int i = 0; i < vecUsers.size(); i++)
            {
                String strName =
(String)vecUsers.elementAt(i);

                UserInfo u =
(UserInfo)m_hashUsers.get(strName);
                if (u == null)
                {
                    u = new UserInfo(strName);
                    if (buf.length() > 0)
                        buf.append(",");
                    buf.append(strName);
                }
                hashNewUsers.put(strName, u);
            }
            // update the current user table to be this
new one
            m_hashUsers = hashNewUsers;
            update();
            return buf.toString();
        }

    public final UserInfo getUserInfo(String
strNick)
```

```
            {
                return (UserInfo)m_hashUsers.get(strNick);
            }

            public final UserInfo getUserInfo(int row)
            {
                return (UserInfo)m_vecKeys.elementAt(row);
            }

            public final Enumeration getUsers()
            {
                return m_vecKeys.elements();
            }

/***************************************************
**************
             * Sets a users specific location
             */
            void setUserLocation(String strNick, String
strUrl)
            {
                UserInfo u = addUser(strNick);
                u.m_strUrl = strUrl;
                update();       // so UI knows location has
changed.
            }

            void setUserActionLocation(String strNick,
String strAction, String strUrl)
            {
                UserInfo u = addUser(strNick);
                u.m_strAction = strAction;
```

```
                    u.m_strUrl = strUrl;
                    update();
                }

                void setUserRealName(String strNick, String
    strRealName)
                {
                    // user must exist to set the name. (User
    messages could come back for ANY channel and
                    // we wouldn't want to add them to our
    channel)
                    UserInfo u = getUserInfo(strNick);
                    if (u != null)
                    {
                        u.setRealName(strRealName);
                        update();        // informing the UI
                    }
                }
            }


    /*******************************************************
    *******************
            *
            */
        /*
        public class MessageListModel extends
    AbstractListModel
        {
            Vector m_vecMsgs = new Vector();
            Vector m_vecListeners = new Vector();
            public void addMsg(String strFrom, String strText)
            {
                m_vecMsgs.addElement(strFrom + ">" + strText);
                int nChanged = m_vecMsgs.size() - 1;
                fireIntervalAdded(this, nChanged, nChanged);
            }

            public int getSize()
            {
                return m_vecMsgs.size();
            }

            public Object getElementAt(int index)
            {
                return m_vecMsgs.elementAt(index);
            }


        }*/



/***********************************************************************
        * Helper function to format the time of a message, if one exists
        */
      static private String getFromAndTime(ChatMessage msg)
      {
          String strFrom = msg.getContributor();
```

```
            String strTime = msg.getDateString();
            if (strTime.length() > 0)
                strFrom = strFrom + "," + strTime;
            return strFrom;
        }

        public class MessageListModel extends
    DefaultListModel
        {
//          Vector m_vecMsgs = new Vector();
//          Vector m_vecListeners = new Vector();

            public void addMsg(String strFrom, String
    strText)
            {
                addElement(strFrom + ">" + strText);
            }

        }
    }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/ChatModel
VerbolListener.java
//+++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */

public interface ChatModelVerbolListener
{
    public void executeVerbol(String strFrom, String
strVerbol, boolean bExecuteWithoutAsking);
    public void textMessage(String strFrom, String
strMessage, int nMsgType);      // a text message to
display
    public void navigateTo(String url, boolean
bForceRefresh);
}
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/ChatPopup
Window.java
//+++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import  sun.audio.*;    //import the sun.audio package
import  java.io.*;
/**
 * A small popup window used for displaying short chat
messages at the bottom
 * of the screen.
 * $Revision: 1.3 $
 * $Date: 2002/02/21 18:12:36 $
 * $Author: markc $
 *
 */

public class ChatPopupWindow implements MouseListener
{
    private JWindow m_window;
    private JLabel m_label1;
    private JLabel m_label2;
    private boolean m_bShown = false;
    private JFrame m_frame = null;

    public ChatPopupWindow()
    {
            // create the controls for the window
            m_window = new JWindow();
            ImageIcon icon = new
    ImageIcon(getClass().getResource("/images/incoming.gif"
    ));
            m_label1 = new JLabel("", icon,
    SwingConstants.LEFT);
            m_label2 = new JLabel("");

            JTextField field = new JTextField();
            Font font = field.getFont();        // use the
    same font that a text field would use by default (i.e.
    not bold)
            m_label1.setOpaque(true);
            m_label1.setBackground(Color.yellow);
            m_label1.setForeground(Color.black);
            m_label1.setFont(font);
            m_label2.setOpaque(true);
            m_label2.setBackground(Color.yellow);
            m_label2.setForeground(Color.black);
            m_label2.setFont(font);
            m_window.getContentPane().add(m_label1,
    BorderLayout.CENTER);
            m_window.getContentPane().add(m_label2,
    BorderLayout.SOUTH);
            m_window.addMouseListener(this);
        }


    /*****************************************************
    *****************
```

```
        * Displays the specfied message (over two lines).
    Window is positioned
        * appropriately to the bottom right of the screen

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

        public void showMessage(String strTitle, String
    strMsg)
        {
            m_label1.setText(strTitle);
            m_label2.setText(strMsg);

            // make the window fit it's text
            m_window.pack();

            // figure out where to display it and make it
    visible
            Dimension sizeScreen =
    Toolkit.getDefaultToolkit().getScreenSize();
            Dimension sizeHint = m_window.getSize();
            m_window.setLocation(sizeScreen.width -
    sizeHint.width - 10, sizeScreen.height -
    sizeHint.height - 40);
            m_window.setVisible(true);
            m_window.toFront();

            if (!m_bShown)
            {
                bing();
                m_bShown = true;
            }
        }
        public void hide()
        {
            m_window.setVisible(false);
            m_bShown = false;
        }

        /****
         * frame to show when this frame gets clicked on
         */
        public void setFrame(JFrame frame)
        {
            m_frame = frame;
        }

        public void mouseClicked(MouseEvent e)
        {
            if (m_frame != null)
            {
                hide();
// In JDK 1.1 we can't do this...argghhh !
                // if iconified - restore the frame
//                if (m_frame.getState() ==
Frame.ICONIFIED)
//                    m_frame.setState(Frame.NORMAL);

                m_frame.setVisible(true);
                m_frame.toFront();
            }
        }

        public void mousePressed(MouseEvent e)
        {
            System.out.println(e);
        }
        public void mouseReleased(MouseEvent e)
        {
        }
        public void mouseEntered(MouseEvent e)
        {
        }
        public void mouseExited(MouseEvent e)
        {
        }


        public void bing()
        {
            //** add this into your application code as
    appropriate
            // Open an input stream  to the audio file.
            try
            {
                //InputStream in = new
    FileInputStream("ding.au");
                InputStream in =
    getClass().getResourceAsStream("/ding.au");
                // Create an AudioStream object from the
    input stream.
                AudioStream as = new AudioStream(in);
                // Use the static class member "player"
    from class AudioPlayer to play
                // clip.
                AudioPlayer.player.start(as);
                // Similarly, to stop the audio.
                //AudioPlayer.player.stop(as);
            }
            catch (Exception e)
            {
                System.out.println(e);
                e.printStackTrace();
            }
        }
    }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/ChatPrope
rtyPage.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

import com.fs.util.*;
import javax.swing.*;
import java.awt.*;
import ChatExtension;
import java.util.Enumeration;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */

public class ChatPropertyPage extends PropertyPage
implements ActionListener
{
    private JCheckBox    m_checkShowVerbol;
    private JList        m_listTrusted;
    private DefaultListModel m_modelTrusted;
    private JButton      m_btnRemove;

    public ChatPropertyPage()
    {
            ChatExtension ext =
    ChatExtension.getInstance();
            this.setLayout(new BoxLayout(this,
    BoxLayout.Y_AXIS));


            m_checkShowVerbol = new JCheckBox("show verbol
    commands");

    m_checkShowVerbol.setSelected(ext.m_bShowVerbol);
            this.add(m_checkShowVerbol);
            m_checkShowVerbol.setAlignmentX(0.0f);

            JPanel paneTrust = new JPanel();
            paneTrust.setLayout(new BoxLayout(paneTrust,
    BoxLayout.X_AXIS));

            m_modelTrusted = new DefaultListModel();
            m_listTrusted = new JList(m_modelTrusted);

            // get the list of trusted users and fill the
    list box.
            Enumeration enum = ext.getTrustedUsers();
            while (enum.hasMoreElements())
            {
                String user = (String) enum.nextElement();
                m_modelTrusted.addElement(user);
            }
            paneTrust.add(new JScrollPane(m_listTrusted));
            m_btnRemove = new JButton("Remove");
            m_btnRemove.addActionListener(this);
            paneTrust.add(m_btnRemove);
```

```
paneTrust.setBorder(BorderFactory.createTitledBorder(Bo
rderFactory.createBevelBorder(10),
                              "Automatically accept Verbol
from"));

            this.add(paneTrust);

        }

        public boolean onApply()
        {
            ChatExtension ext =
ChatExtension.getInstance();

            ext.m_bShowVerbol =
m_checkShowVerbol.isSelected();

            ext.setTrustedUsers(m_modelTrusted.elements());

            return true;            // no validation.
        }



        public void actionPerformed(ActionEvent e)
        {
            Object source = e.getSource();
            if (source == m_btnRemove)
            {
                onRemoveTrust();
            }
        }
        private void onRemoveTrust()
        {
            int nSel = m_listTrusted.getSelectedIndex();
            if (nSel >= 0)
                m_modelTrusted.remove(nSel);
        }

        protected void onHelp()
        {

        getSheet().showHelp("\\docs\\controller\\xh\\xh_co
nf_chat.htm");
        }

}
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/ChatServe
rResolver.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

/**
 * This class resolves the ChatServer name for a given
ibook and host.
 * Currently does this by looking for a specific file
in the ibook chat room
 * @author $Author: markc $
 * @version $Revision: 1.2 $
 */
// $Date: 2002/04/12 14:48:00 $
import com.fs.chatclient.*;
import java.util.*;
import java.net.*;
import java.io.*;


public class ChatServerResolver
{
    private Hashtable g_hashHosts = new Hashtable();

    public ChatServerResolver()
    {
    }


/****************************************************
*****************
```

```
         * Clears any cached knowledge about the mappings
     of ibook servers to sites, etc
         */
        public void clear()
        {
            g_hashHosts.clear();
        }


    /*******************************************************
    *****************
         * Returns the server for a specific ibook host,
    ibook path. Username and password
         * are required IF authorization is required to
    access pages on the site (as is the
         * case in ibook sites)

    *******************************************************
    ***************/
        public ServerConnectionInfo getServer(String
    strHost, String strPath, String strUser, String
    strPassword)
        {
            // first look in the path specified, then in
    the all parent directories
            ServerConnectionInfo info = null;
            while (true)
            {
                info = searchServer(strHost + strPath,
    strUser, strPassword);
                if (info != null)
                    return info;            // found
                int nSlash = strPath.lastIndexOf('/');
                if (nSlash < 0)
                    break;          // no other paths to
try
                strPath = strPath.substring(0, nSlash);
            }

            // look in the root of the site.
            info = searchServer(strHost, strUser,
strPassword);
            return info;
        }


        private ServerConnectionInfo searchServer(String
strUrl, String strUser, String strPassword)
        {
            // first look to see if we've got information
for this server already - we might have
            ServerConnectionInfo info =
(ServerConnectionInfo)g_hashHosts.get(strUrl);
            if (info != null)
                return info;

            try
            {
                URL url = new URL("http://" + strUrl +
"/ibook.props");
                Properties props = getProperties(url,
strUser, strPassword);    // need to know about
authentication
                info = new ServerConnectionInfo();
                info.m_strServer =
props.getProperty("irc.server");
                if (info.m_strServer == null)
                    return null;                        // no
irc.server entry--> no chat server setting

                String strPort =
props.getProperty("irc.port");
                info.m_nPort = 6667;
                if (strPort != null)
                {
                    try
                    {
                        info.m_nPort =
Integer.parseInt(strPort);

                    }
                    catch (NumberFormatException e)
                    {
                    }
                }
                g_hashHosts.put(strUrl, info);
            }
            catch (IOException e)
            {
                e.printStackTrace();
                return null;
            }
            return info;

        }
```

```
    static public void main (String args[])
    {
        ChatServerResolver csr = new
ChatServerResolver();
        ServerConnectionInfo info =
csr.getServer("http://www.familysystems.org",

"/FamilyStaff/MarkConway",

"markc", "match+made");
        System.out.println("info=" + info);

        try
        {
            URL url = new
URL("http://www.familysystems.org/FamilyStaff/MarkConwa
y/ibook.props");
            dumpURL(url);
            Properties props = getProperties(url,
"markc", "match+made");
            props.list(System.out);

        }
        catch (Exception e)
        {
            e.printStackTrace();
        }

    }

    static public void dumpURL(URL url)
    {
        System.out.println("url=" + url);
```

```
            //System.out.println("user=" +
    url.getUserInfo());
            System.out.println("host=" + url.getHost());
            //System.out.println("path=" + url.getPath());
            System.out.println("file=" + url.getFile());
        }


        static public void dumpStream(InputStream is)
    throws IOException
        {
            PrintStream pw = System.out;
            BufferedReader in = new BufferedReader (new
    InputStreamReader (is));
            String line;
            while ((line = in.readLine()) != null)
            {
                pw.println (line);
            }
        }


        static public InputStream getHttpStream (URL url,
    String user, String password)
                throws IOException
        {
            String userPassword = user + ":" + password;
            String encoding = new
    sun.misc.BASE64Encoder().encode
    (userPassword.getBytes());
            URLConnection uc = url.openConnection();
            uc.setRequestProperty  ("Authorization", "Basic
    " + encoding);
            InputStream content =
    (InputStream)uc.getInputStream();
            return content;
        }


        static public Properties getProperties(URL url,
    String user, String password)
        {
            Properties p = new Properties();
            try
            {
                InputStream in = getHttpStream(url, user,
    password);
                p.load(in);
                in.close();
            }
            catch (Exception e)
            {
            }
            return p;
        }
    }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/ChatTextP
ane.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

import javax.swing.*;
import javax.swing.text.*;
import java.awt.*;                //for layout managers
import java.awt.event.*;          //for action and window
events
import java.util.Hashtable;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */

public class ChatTextPane extends JTextPane implements
MouseListener, MouseMotionListener
{
    private static Hashtable g_hashIconMap = null;

    // mappings for "chaticons"
    private static String[][] ICON_MAP =
    {
        { ":D", "icon_biggrin.gif"},
        { ":-D", "icon_biggrin.gif"},
            { ":grin:", "icon_biggrin.gif"},

            { ":)", "icon_smile.gif" },
            { ":-)", "icon_smile.gif" },
            { ":smile:", "icon_smile.gif" },

            { ":(", "icon_frown.gif" },
            { ":-(", "icon_frown.gif" },
            { ":sad:", "icon_frown.gif" },

//          { ":o", "icon_eek.gif" },      - clashes with
the start of :oops:
            { ":-o", "icon_eek.gif" },
            { ":eek:", "icon_eek.gif" },

            { ":-?", "icon_confused.gif" },
            { ":???:", "icon_confused.gif" },

            { "8-)", "icon_cool.gif" },
            { ":cool:", "icon_cool.gif" },

            { "B-)", "icon_toocool.gif" },
            { ":toocool:", "icon_toocool.gif" },

//          { ":lol:", "icon_lol.gif"},           this
GIF doesn't work for some reason

            { ":-x", "icon_mad.gif"},
            { ":mad:", "icon_mad.gif"},

            { ":-p", "icon_razz.gif"},
            { ":razz:", "icon_razz.gif"},
            { ":oops:", "icon_redface.gif"},

            { ":cry:", "icon_cry.gif"},

            { ":roll:", "icon_rolleyes.gif"},

            { ";-)", "icon_wink.gif" },
            { ";)", "icon_wink.gif" },
            { ":wink:", "icon_wink.gif" },

            { ":cat:", "icon_cat.gif" },
            { ":wave:", "icon_wave.gif" },

            { ":nono:", "icon_nono.gif" },
            { ":jester:", "icon_jester.gif" },
            { ":crazy:", "icon_crazy.gif" },
            { ":contract:", "icon_makeadeal.gif" },
            { ":evil:", "icon_evil.gif" },
        };

        public ChatTextPane()
        {
            initStyles();
            setEditable(false);
//            this.addMouseListener(this);
            this.addMouseMotionListener(this);
            StandardEditActions.addStandardMenu(this);   //
    add Cut/copy/paste popup
        }

        static public String[][] getIconMap()
        {
            return ICON_MAP;
        }


    /*****************************************************
    ********************
         * finds the location of an icon within the
    ICON_MAP
         *

    *****************************************************
    ******************/
        private static int findIcon(String str)
        {
            for (int i = 0; i < ICON_MAP.length; i ++)
            {
                if (str.startsWith(ICON_MAP[i][0]))
                {
                    return i;
                }
            }
            return -1;
        }

        void initStyles()
        {
            //Initialize some styles.
            Style def =
    StyleContext.getDefaultStyleContext().

    getStyle(StyleContext.DEFAULT_STYLE);

            Style regular = this.addStyle("regular", def);
            StyleConstants.setFontFamily(regular, "Arial");
            StyleConstants.setForeground(regular,
    Color.blue);

            Style styleFrom = this.addStyle("from",
    regular);
            StyleConstants.setBold(styleFrom, true);
            StyleConstants.setForeground(styleFrom,
    Color.blue);

            Style s = this.addStyle("verbol", regular);
            StyleConstants.setForeground(s,
    Color.darkGray);

            s = this.addStyle("normal", regular);
            StyleConstants.setForeground(styleFrom,
    Color.blue);

            s = this.addStyle("url", regular);
            StyleConstants.setItalic(s, true);
            StyleConstants.setUnderline(s, true);

            s = this.addStyle("server", regular);
            StyleConstants.setItalic(s, true);
            StyleConstants.setForeground(s,
    Color.darkGray);
        }


    /*******************************************************
    *******************
         * Adds a message to this window
         */
    public void addMessage(String strFrom, String
strMsg)
    {
        Document doc = this.getDocument();

        try
        {
            doc.insertString(doc.getLength(), strFrom +
"> ",
                                    this.getStyle("from"));

            String strDefStyle = "normal";
            if (strFrom.length() == 0)
                strDefStyle = "server";

            parseText(strMsg, strDefStyle);

            doc.insertString(doc.getLength(), "\n",
this.getStyle(StyleContext.DEFAULT_STYLE));
        }
        catch (BadLocationException ble)
        {
            System.err.println("Couldn't insert initial
text.");
        }
    }



/*******************************************************
********************
     * Guess if it's a URL
     */
    static public boolean isURL(String strTest)
    {
        if (strTest.startsWith("http:")
            || strTest.startsWith("file:")
            || strTest.startsWith("www."))
            return true;

        int nEnd = strTest.indexOf('/');
        if (nEnd > 0)
            strTest = strTest.substring(0, nEnd);

        if ( strTest.endsWith(".net")
            || strTest.endsWith(".com")
            || strTest.endsWith(".co.uk"))
            return true;

        return false;
    }


    private void parseText(String strMsg, String
strDefStyle)
    {
        // parsing strMsg for URLS !
        int nPrev = 0;      // place we last output
attribs for
        int nSt = 0;
        nSt = skipPunc(strMsg, nSt);
        if
(strMsg.substring(nSt).startsWith("VERBOL!"))
            strDefStyle = "verbol";
        Document doc = getDocument();
```

```
            while (nSt < strMsg.length())
            {
                int nEnd = skipNonPunc(strMsg, nSt);
                String strTest = strMsg.substring(nSt,
  nEnd);

                int nIcon = findIcon(strTest);
                if (nIcon != -1)
                {
                    String strImage = ICON_MAP[nIcon][1];
                    Style s = this.getStyle(strImage);
                    if (s == null)
                    {
                        s = addStyle(strImage,
  getStyle("regular"));
                        StyleConstants.setIcon(s, new
  ImageIcon(getClass().getResource("/images/" +
  strImage)));
                        StyleConstants.setAlignment(s,
  StyleConstants.ALIGN_CENTER);
                    }
                    flushText(strMsg, nPrev, nSt,
  strDefStyle);
                    nEnd = nSt +
  ICON_MAP[nIcon][0].length();      // only skip the
  characters recognised !
                    nPrev = nEnd;
                    try
                    {
                        doc.insertString(doc.getLength(), "
  ", s);      // insert the icon style
                    }
                    catch (BadLocationException ble)
                    {
                    }
                }
                else if (isURL(strTest))
                {
                    // it's a URL - add all previous text
                    flushText(strMsg, nPrev, nSt,
    strDefStyle);
                    flushText(strMsg, nSt, nEnd,  "url");
                    nPrev = nEnd;
                }
                nSt = skipPunc(strMsg, nEnd);
            }
            flushText(strMsg, nPrev, nSt, strDefStyle);
        }

        final private void flushText(String strMsg, int
    nSt, int nEnd, String strStyle)
        {
            if (nEnd > nSt)
            {
                Document doc = this.getDocument();
                try
                {
                    String strText = strMsg.substring(nSt,
    nEnd);
                    doc.insertString(doc.getLength(),
    strText, this.getStyle(strStyle));
                    //System.out.println("added " + strText
    + "(" + strStyle + ")" );
                }
                catch (BadLocationException ble)
                {
                    System.err.println("Couldn't insert
    initial text.");
                }
            }
        }

        static final private boolean isPunc(char c)
        {
            return (c == ' ' || c== ',' || c == '!');
        }

        static final private int skipNonPunc(String strMsg,
    int nSt)
        {
            while (nSt < strMsg.length())
            {
                char ch = strMsg.charAt(nSt);
                if (isPunc(ch))
                    break;
                nSt++;
            }
            return nSt;
        }

        static final private int skipPunc(String strMsg,
    int nSt)
        {
            while (nSt < strMsg.length())
            {
                char ch = strMsg.charAt(nSt);
                if (!isPunc(ch))
                    break;
                nSt++;
            }
            return nSt;
        }

        public void iterateDoc()
        {
            Document doc = this.getDocument();
            Element elt = doc.getDefaultRootElement();
            iterateElement(elt, 0);
        }

        void iterateElement(Element elt, int nLevel)
        {
            Document doc = this.getDocument();
            int nSt = elt.getStartOffset();
            int nEnd = elt.getEndOffset();
            try
            {
                String strText = doc.getText(nSt, nEnd -
    nSt);
                strText = strText.replace('\n', '~');
                System.out.println("........".substring(0,
    nLevel) + " st="
                        + nSt +  " end=" + nEnd
                        + " text=" + strText);
            }
            catch (BadLocationException ble)
            {
            }
            for (int i = 0; i < elt.getElementCount(); i++)
            {
                Element e2 = elt.getElement(i);
                iterateElement(e2, nLevel + 1);
            }

        }


/***************************************************************************
     * Returns the text of the element at the specified
co-ordinates (mouse click)
     */
    public String getTextAtPoint(Point pt)
    {
        int nOffs = this.viewToModel(pt);
        Document doc = this.getDocument();
        Element elt =
findLeafElementAtPos(doc.getDefaultRootElement(),
nOffs);
        int nSt = elt.getStartOffset();
        int nEnd = elt.getEndOffset();
        String strText = null;
        try
        {
            strText = doc.getText(nSt, nEnd - nSt);
        }
        catch (BadLocationException ble)
        {
            strText = "";
        }
        return strText;
    }
```

```
        public Element findLeafElementAtPos(int nPos)
        {
            return
    findLeafElementAtPos(this.getDocument().getDefaultRootE
    lement(), nPos);
        }

        Element findLeafElementAtPos(Element elt, int nPos)
        {
            while (!elt.isLeaf())
            {
                elt =
    elt.getElement(elt.getElementIndex(nPos));
            }
            return elt;
        }

        public void mouseClicked(MouseEvent parm1)
        {
            if (parm1.getClickCount() != 2)
                return;
            Point pt = parm1.getPoint();
            int nOffs = this.viewToModel(pt);

            // where is the cursor position ?
            //int nOffs = this.getCaretPosition();

            Document doc = this.getDocument();
            Element elt =
    findLeafElementAtPos(doc.getDefaultRootElement(),
    nOffs);
            int nSt = elt.getStartOffset();
            int nEnd = elt.getEndOffset();
            try
            {
                String strText = doc.getText(nSt, nEnd -
    nSt);
                System.out.println("double click " + " st="
                        + nSt +  " end=" + nEnd
                        + " text=" + strText);
            }
            catch (BadLocationException ble)
            {
            }
            // tell the parent model to navigate here....

        }

        public void mousePressed(MouseEvent parm1)
        {
        }
        public void mouseReleased(MouseEvent parm1)
        {
        }
        public void mouseEntered(MouseEvent parm1)
        {
        }
        public void mouseExited(MouseEvent parm1)
        {
        }

        public void mouseDragged(MouseEvent parm1)
        {
        }

        public void mouseMoved(MouseEvent parm1)
        {
            Point pt = parm1.getPoint();
            String strText = getTextAtPoint(pt);
            setCursor(isURL(strText) ?
Cursor.getPredefinedCursor(Cursor.HAND_CURSOR)
                                            : null);

        }


/******************************************************
********************
        * Scrolls to ensure the end of the document is
visible

******************************************************
******************/
    public void scrollToEnd()
    {
        Document doc = getDocument();
        // somehow on MS JVM we can end up with doc ==
null
        if (doc != null)
            scrollToPos(doc.getLength());
    }



/******************************************************
********************
        * Scrolls to ensure the specified position in the
document is visible
```

```
*******************************************************
*****************/
    public void scrollToPos(int nPos)
    {
        try
        {
            Rectangle rect = this.modelToView(nPos);
            if (rect != null)
                scrollRectToVisible(rect);
        }
        catch (BadLocationException e)
        {
            e.printStackTrace();
        }
    }



/*******************************************************
******************
     * Clears the window

*******************************************************
****************/
    public void clear()
    {
        Document doc = getDocument();
        try
        {
            doc.remove(0, doc.getLength());
        }
```

```
            catch (BadLocationException e)  // shouldn't
    happen !
            {
            }
        }
    }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/ChatVerbo
l.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

/**
 * The class encapsulates some routines for VERBOL in
the chat messages
 * Meant to provide a common reference point.
 * @author $Author: markc $
 * @version $Revision: 1.3 $
 *
 */
// $Date: 2002/04/12 14:48:00 $

public class ChatVerbol
{

    static public final String VERBOL_INDICATOR =
"VERBOL!";


    static public final String REVISING_VERB =
"REVISING";
    static public final String BROWSES_VERB =
"BROWSES";
    static public final String COMPLETEDREVISION_VERB =
"COMPLETEDREVISION";
    static public final String REVISED_VERB =
"REVISED";

    static public final String[] VERBS =
        {
            BROWSES_VERB,
            REVISING_VERB,
            COMPLETEDREVISION_VERB,
            REVISED_VERB
        };



    /*******************************************************
*****************
         * Return true if the verb is a "special" verb that
    the Chat extension
         * would normally handle
         */
        static public boolean isSpecialVerb(String verb)
        {
            for  (int i = 0; i < VERBS.length; i++)
            {
                if (VERBS[i].equalsIgnoreCase(verb))
                    return true;
            }
            return false;
        }



        public ChatVerbol()
        {
        }
    }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/Connectio
nDialog.java
//+++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

import javax.swing.*;
import javax.swing.text.JTextComponent;
import java.awt.*;             //for layout managers
import java.awt.event.*;       //for action and window
events
import com.fs.chatclient.ServerConnectionInfo;
import ChatInstance;
import ChatExtension;
/**
 * Implements a dialog for the user to enter connection
info into for a chat
 * connection
 * $Date: 2002/04/20 22:36:22 $
 * $Author: markc $
 * $Revision: 1.9 $
 */

public class ConnectionDialog extends JDialog
{
    private VTextField     m_editServer;
    private VTextField     m_editPort;
    private JPasswordField m_editPass;
    private JTextField     m_editNick;
    private JTextField     m_editUser;
    private JTextField     m_editChannel;
    private JTextField     m_editRealName;
    private boolean         m_bConnectPressed = false;

    private ServerConnectionInfo  m_info;           //
connection on output
    private String          m_strChannel;           //
channel to talk on

    public ConnectionDialog()
    {
        super();
        setTitle("Connection Details");

        //Create the fields
        m_editRealName = new VTextField(40);
        m_editServer = new VTextField(40);
        m_editPort = new VTextField(5);
        m_editNick = new VTextField(20);
        m_editUser = new VTextField(20);
        m_editPass = new JPasswordField(20);
        m_editChannel = new VTextField(40);

        //Create a panel for the labels
        JPanel textControlsPane = new JPanel();

textControlsPane.setBorder(BorderFactory.createEmptyBor
der(10, 0, 10, 0));
        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints c = new
GridBagConstraints();

        textControlsPane.setLayout(gridbag);

        JTextField[] textFields =
            {
                    m_editServer,
                    m_editPort,
                    m_editRealName,
                    m_editNick,
                    m_editUser,
                    m_editPass,
                    m_editChannel
            };

            String[] arrLabels = { "Server:", "Port:",
    "Real name:", "Nickname:", "User:", "Password:", "Chat
    Room:"};

            addLabelTextRows(arrLabels, textFields,
    gridbag, textControlsPane);

            Container content = getContentPane();
            content.add(textControlsPane,
    BorderLayout.CENTER);
            content.add (new JLabel("Please enter details
    for the chat room you wish to join"),
    BorderLayout.NORTH);
            content.add(createButtonRow(),
    BorderLayout.SOUTH);

            pack();
        }

        public boolean wasConnectPressed()
        {
            return m_bConnectPressed;
        }
```

```
/****************************************************
*******************
     * Create the row of Connect, Cancel, Help buttons
     */
    private JPanel createButtonRow()
    {
        JPanel buttonPane = new JPanel();
          JButton btnOK = new JButton("Connect");
          btnOK.addActionListener(new ActionListener()
               {
                    public void
actionPerformed(ActionEvent e)
                    {
                    onConnect();
                    }
               });
        this.getRootPane().setDefaultButton(btnOK); //
set the Connect as the default button

          JButton btnCancel = new JButton("Cancel");
          btnCancel.addActionListener(new
ActionListener()
               {
                    public void
actionPerformed(ActionEvent e)
                    {
                         hide();
                    }
               });

          JButton btnHelp = new JButton("Help");
```

```
            btnHelp.addActionListener(new
ActionListener()
                {
                    public void
actionPerformed(ActionEvent e)
                    {
                        onHelp();
                    }
                });

            // Organized in a row, with the Help button
aligned to the right
            buttonPane.setLayout(new
BoxLayout(buttonPane, BoxLayout.X_AXIS));

buttonPane.setBorder(BorderFactory.createEmptyBorder(10
, 10, 10, 10));
        buttonPane.add(Box.createHorizontalGlue());
        buttonPane.add(btnOK);
        buttonPane.add(Box.createRigidArea(new
Dimension(10, 0)));
        buttonPane.add(btnCancel);
        buttonPane.add(Box.createHorizontalGlue());
        buttonPane.add(btnHelp);

        return buttonPane;
    }


    void onHelp()
    {
    }
    public ServerConnectionInfo getConnectionInfo()
    {
        return getInfoFromControls();
    }

    public void setConnectionInfo(ServerConnectionInfo
info)
    {
        setControlsFromInfo(info);
    }

    private void
setControlsFromInfo(ServerConnectionInfo info)
    {
        m_editServer.setText(info.m_strServer);

m_editPort.setText(Integer.toString(info.m_nPort));
        m_editNick.setText(info.m_strNick);
        m_editUser.setText(info.m_strUser);
        m_editPass.setText(info.m_strPass);
        m_editRealName.setText(info.m_strRealName);
    }


    private ServerConnectionInfo getInfoFromControls()
    {
        ServerConnectionInfo info = new
ServerConnectionInfo();
        info.m_strServer =
m_editServer.getText().trim();
        try
```

```
            {
                info.m_nPort =
    Integer.parseInt(m_editPort.getText().trim()) ;
            }
            catch (NumberFormatException e)
            {
                info.m_nPort =0;
            }
            info.m_strUser = m_editUser.getText().trim();
            info.m_strPass = m_editPass.getText().trim();
            info.m_strNick = m_editNick.getText().trim();
            info.m_strRealName =
    m_editRealName.getText().trim();
            return info;
        }


        private void reportError(JComponent comp, String
    message)
        {
            JOptionPane.showMessageDialog(this, message,
    getTitle(), JOptionPane.ERROR_MESSAGE);
            if (comp != null)
                comp.requestFocus();    // put focus in the
    correct place
        }


        private static boolean checkLegalChars(String str)
        {
            for (int i = 0; i < str.length(); i++)
            {
                char ch = str.charAt(i);
                if (!Character.isLetterOrDigit(ch)
                        && ch != '_' && ch != '-')
                    return false;   // illegal character
            }
            return true;
        }


        private void onConnect()
        {
            m_info = getInfoFromControls();
            m_strChannel = m_editChannel.getText().trim();

            // validation stuff...
            if (m_info.m_strServer.length() == 0)
            {
                reportError(m_editServer, "Please specify a
    server");
                return;
            }

            if (m_info.m_strNick.length() == 0)
            {
                reportError(m_editNick, "Please specify a
    nickname");
                return;
            }
            if (!checkLegalChars(m_info.m_strNick))
            {
                reportError(m_editNick, "Nickname should
    only contain A-Z, 0-9 and _, -");
                return;
            }
            if (m_info.m_nPort == 0)
            {
                reportError(m_editPort, "Please specify a
port. The default port is 6667");
                return;
            }

            if (m_strChannel.length() == 0)
            {
                reportError(m_editChannel, "Please specify
a chat room");
                return;
            }
            if (!checkLegalChars(m_strChannel))
            {
                reportError(m_editChannel, "Chat room
should only contain A-Z, 0-9 and _, -");
                return;
            }

            ChatExtension ext =
ChatExtension.getInstance();
            ext.setConfigChatChannel(getChannel());
            ext.setConfigChatInfo(getConnectionInfo());
            ChatInstance instance =
ChatInstance.newInstance(ChatExtension.getInstance(),
getConnectionInfo(), getChannel());
            ext.updateControllerTitle();
            hide();
            m_bConnectPressed = true;

            //dispose();
```

```
        }


        public String getChannel()
        {
            return m_editChannel.getText();
        }

        public void setChannel(String str)
        {
            m_editChannel.setText(str);
        }


    /*****************************************************
*******************
        * Helper function to label fields in a table in a
GridBag
        */
        private void addLabelTextRows(String[] labels,
                                    JTextField[]
textFields,
                                    GridBagLayout
gridbag,
                                    Container container)
        {
            GridBagConstraints c = new
GridBagConstraints();
            c.anchor = GridBagConstraints.EAST;
            int numLabels = labels.length;

            for (int i = 0; i < numLabels; i++)
            {
                c.gridy = i;
                c.gridx = 0;
                c.gridwidth = 1;
                c.gridheight = 1;
                c.fill = GridBagConstraints.NONE;
//reset to default
                c.weightx = 0.0;
//reset to default
                JLabel label = new JLabel(labels[i]);
                label.setLabelFor(textFields[i]);
                gridbag.setConstraints(label, c);
                container.add(label);

                c.gridx = 1;
                //c.gridwidth =
GridBagConstraints.REMAINDER;     //end row
                c.fill = GridBagConstraints.HORIZONTAL;
                c.weightx = 1.0;
                gridbag.setConstraints(textFields[i], c);
                container.add(textFields[i]);
            }

            /*
             * experimental code to add a helper text area
to the right of the fields
             * might use in future.
            c.gridwidth = 1;
            c.gridheight = numLabels;
            c.gridx = 2;
            c.gridy = 0;
            c.weightx = 0.0;
            c.fill = GridBagConstraints.VERTICAL;
            JTextArea text = new JTextArea("help text");
```

```
                text.setBackground(Color.yellow);
                text.setLineWrap(true);
                text.setPreferredSize(new Dimension(100, 50));
                gridbag.setConstraints(text, c);
                gridbag.setConstraints(text, c);
                container.add(text);
                */
        }

}
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/DirtyComm
s.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */
import java.io.*;
import java.net.*;
import ChatExtension;

public class DirtyComms extends Thread
{
    private ServerSocket m_sockMain;          // main
server socket
    private int          m_nPort;             // port to
listen on
    private boolean      m_bStopped;
    private ChatExtension   m_ext;            // the
extension we belong to

    public DirtyComms(ChatExtension ext)
    {
        m_ext = ext;
        m_nPort = 7000;
        }

        public void requestStop()
        {
            m_bStopped = true;
        }

        public boolean isStopped()
        {
            return m_bStopped;
        }

        public void run()
        {
            log("Listening on port " + m_nPort);
            m_bStopped = false;
            //setName("ListenOnPort" + m_nPort);
    // set name of thread to something more obvious !
            try
            {
                m_sockMain = new ServerSocket(m_nPort);
                m_sockMain.setSoTimeout(10 * 1000);  // set
    the timeout so we check whether to stop or not !
            }
            catch (IOException e)
            {
                log(e, "DirtyComms unable to bind to
    socket");
                return;          // nothing we can do -
    there are probably 2 chat clients running
            }
```

```
                while (!m_bStopped)
                {
                    try
                    {
                        Socket sockNew = m_sockMain.accept();

                        // received an incoming connection -
allocate a client connection to handle it
                        new IncomingThread(sockNew);
                    }
                    catch (InterruptedIOException e)
                    {
                        // swallow silently - we're just going
round the SoTimeout loop
                    }
                    catch (Exception e)
                    {
                        log(e, "MainSocket - listen");
                    }
                }
                try
                {
                    m_sockMain.close();
                }
                catch (IOException e)
                {
                    log(e, "Exception closing DirtyComms
socket");
                }
            }
```

```
/*******************************************************
*****************
     * Process an incoming message
     */
    private void processMessage(String message)
    {
        // what is the message we are processing ?
        StringTokenizer2 tok = new
StringTokenizer2(message, " ");
        if (!tok.hasMoreTokens())
            return;     // ignore it if no first token

        String command = tok.nextToken();
        if (command.equals("SEND"))

m_ext.sendMessageToAllChannels(tok.getRemainder());

    }


    private void log(String s)
    {
        System.out.println(s);
    }

    private static void log(Exception e, String s)
    {
        System.out.println(s);
        e.printStackTrace();
    }
```

```
/*****************************************************
****************
      * Very simple thread to read from the incoming
socket stream. Just passes the results
      * through to the callback in the parent.
      */
     private class IncomingThread extends Thread
     {
          BufferedReader m_receive;
          IncomingThread(Socket sock) throws IOException
          {
               m_receive = new BufferedReader(new
InputStreamReader(sock.getInputStream()));
               setName("DirtyCommsReceiver");
               start();
          }


          public void run()
          {
               log("IncomingThread starting");

               while (!isStopped())
               {
                    try
                    {
                         String str = m_receive.readLine();
                         // getting NULL normally means the
socket is finished
                         if (str == null)
                              throw new
IOException("readLine() returned null");
```

```
                                  try
                                  {
                                      processMessage(str);
                                  }
                                  catch (Throwable e)
                                  {

//ChatInstance.reportException(e);  // an exception
here should NOT kill us - report it, because it needs
investigating !
                                  }
                              }
                              catch (InterruptedIOException e)
                              {
                                  // silently swallow this message -
done so we check the stopped flag
                                  // every so often
                              }
                              catch (Exception e)
                              {
                                  // any error - break the connection
                                  break;
                              }
                          }
                          try
                          {
                              m_receive.close();
                          }
                          catch (IOException e)
                          {
                              log(e, "error closing incoming
thread");
```

```
                    }
                    System.out.println("DirtyCommsThread
ended");
                }
            }

        }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/DlgHistor
y.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

import javax.swing.*;
import java.awt.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */

public class DlgHistory extends JPanel
{
    WholeNumberField m_textMins;

    public DlgHistory()
    {
        super();
        this.setLayout(new BorderLayout());

        JPanel panel = this;
        panel.setLayout(new BoxLayout(panel,
BoxLayout.X_AXIS));
        panel.add(new JLabel("Show all messages in the
last"));
```

```
            m_textMins = new WholeNumberField(10, 5);
            panel.add(m_textMins);
            panel.add(new JLabel("minutes"));

            //this.add(panel, BorderLayout.CENTER);
        }

        public int getMinutes()
        {
            return m_textMins.getValue();
        }



    }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/HashTable
Model.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author $Author: markc $
 * @version $Revision: 1.2 $
 * THIS CLASS NO LONGER USED
 */
// $Date: 2002/04/15 08:53:01 $
import java.util.*;
import javax.swing.table.*;
import javax.swing.event.TableModelListener;

public class HashTableModel extends AbstractTableModel
implements TableModel
{
    private Hashtable m_hash = new Hashtable();
// hash table that drives the model.
    private Vector m_vecKeys = new Vector();
    private String[] m_strColumnName = { "user",
"location" };
    private Vector m_vecListeners = new Vector();

    public HashTableModel()
    {
    }

    public HashTableModel(Hashtable hash)
    {
        setHashtable(hash);
    }

    public void setHashtable(Hashtable hash)
    {
        m_hash = hash;
        m_vecKeys = new Vector(m_hash.size());
        Enumeration enum = hash.keys();
        while (enum.hasMoreElements())
        {
            m_vecKeys.addElement(enum.nextElement());
        }
        fireTableDataChanged();
    }


    public int getRowCount()
    {
        return m_hash.size();
    }

    public int getColumnCount()
    {
        return 2;    // always 2
    }

    public String getColumnName(int columnIndex)
    {
        return m_strColumnName[columnIndex];
    }

    public Class getColumnClass(int columnIndex)
    {
        return new String().getClass();
    }

    /*
    public boolean isCellEditable(int rowIndex, int
columnIndex)
    {
        return false;
    }*/

    public Object getValueAt(int rowIndex, int
columnIndex)
    {
        Object key = m_vecKeys.elementAt(rowIndex);
        if (columnIndex == 0)
            return key;
        else
            return m_hash.get(key);
    }

    /*
    public void setValueAt(Object aValue, int rowIndex,
int columnIndex)
    {
        throw new
java.lang.UnsupportedOperationException("Method
setValueAt() not yet implemented.");
    }
    */
```

```
        /*
        public void
    addTableModelListener(TableModelListener l)
        {
            m_vecListeners.addElement(l);
        }

        public void
    removeTableModelListener(TableModelListener l)
        {
            m_vecListeners.removeElement(l);
        }*/
    }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/Quicksort
Algorithm.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

/**
 * An implementation of the QuickSort algorithm,
because JDK 1.1 doesn't have
 * one.
 * @author $Author: markc $
 * @version $Revision: 1.3 $
 * $Date: 2002/04/15 08:53:01 $
 */

public class QuicksortAlgorithm
{
    interface Comparator
    {
        public int compare(Object o1, Object o2);
    }



    static public class CompareAsStrings implements
Comparator
    {
        public int compare(Object o1, Object o2)
        {
            if (o1 == null)
                return -1;      // null is less than
anything
            if (o2 == null)
```

```
                return +1;      // anything is more
than null
            return
o1.toString().compareTo(o2.toString());
        }
    }

    static public void main(String args[])
    {
        String [] arrSort = { "brian", "mark",
"charly", "fred" };
        dump(arrSort);
        Comparator c = new CompareAsStrings();
        sort(arrSort, c);
        testOrder(arrSort, c);
        dump(arrSort);
    }


    static public void dump(Object[]'arr)
    {
        for (int i = 0; i < arr.length; i++)
        {
            System.out.println(i + " - " + arr[i]);
        }
    }

    static public void testOrder(Object[] arr,
Comparator c)
    {
        for (int i = 0; i < arr.length - 1; i++)
        {
            if (c.compare(arr[i], arr[i + 1]) > 0)
```

```
                {
                    System.out.println("Compare test failed
    - " + arr[i] + " vs " + arr[i + 1]);
                }
            }
    }



        public QuicksortAlgorithm()
        {
        }

       /** This is a generic version of C.A.R Hoare's Quick
    Sort
         * algorithm.  This will handle arrays that are
    already
         * sorted, and arrays with duplicate keys.<BR>
         *
         * If you think of a one dimensional array as going
    from
         * the lowest index on the left to the highest index
    on the right
         * then the parameters to this function are lowest
    index or
         * left and highest index or right.  The first time
    you call
         * this function it will be with the parameters 0,
    a.length - 1.
         *
         * @param a       an integer array
         * @param lo0     left boundary of array partition
         * @param hi0     right boundary of array partition
        */
       static void QuickSort(Object a[], int lo0, int hi0,
    Comparator c)
       {
          int lo = lo0;
          int hi = hi0;
          Object mid;

          if ( hi0 > lo0)
          {

             /* Arbitrarily establishing partition element
   as the midpoint of
              * the array.
              */
             mid = a[ ( lo0 + hi0 ) / 2 ];

             // loop through the array until indices cross
             while( lo <= hi )
             {
                /* find the first element that is greater
  than or equal to
                 * the partition element starting from the
  left Index.
                 */
                while( ( lo < hi0 ) && ( c.compare(a[lo],
  mid) < 0 ) )
                   ++lo;

                /* find an element that is smaller than or
  equal to
                 * the partition element starting from the
  right Index.
```

```
                    */
                   while( ( hi > lo0 ) && ( c.compare (a[hi],
mid) > 0 ) )
                      --hi;

                   // if the indexes have not crossed, swap
                   if( lo <= hi )
                   {
                      swap(a, lo, hi);
                      ++lo;
                      --hi;
                   }
                }

                /* If the right index has not reached the left
side of array
                 * must now sort the left partition.
                 */
                if( lo0 < hi )
                   QuickSort( a, lo0, hi, c );

                /* If the left index has not reached the right
side of array
                 * must now sort the right partition.
                 */
                if( lo < hi0 )
                   QuickSort( a, lo, hi0, c );

             }
       }

       static private void swap(Object a[], int i, int j)
       {
            Object T = a[i];
            a[i] = a[j];
            a[j] = T;
        }



/*****************************************************
*****************
     * Sorts an array of objects according to the
Comparator function supplied
     */
    static public void sort(Object a[], Comparator c)
    {
        QuickSort(a, 0, a.length - 1, c);
    }



/*****************************************************
*****************
     * Sorts an array of objects according to their
natural toString() order i.e.
     * each objects toString() function is called and
they are compared using that
     */
    static public void sort(Object a[])
    {
      QuickSort(a, 0, a.length - 1, new
CompareAsStrings());

    }
```

```
}
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/StandardE
ditActions.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;
/**
 * Assists in the implementation of standard
cut/copy/paste options for a class.
 * Each instance of this class represents the necessary
customizations for a
 * particular text component. These are accessed via a
hashtable, so the popup
 * menu, actions, etc are shared between TextComponents
of the same class. This
 * is to reduce the overhead, as in practice you want
most (all) TextComponents to
 * have this behaviour
 *
 * @author $Author: markc $
 */
// $Date: 2002/03/19 17:23:45 $
// $Revision: 1.1 $

import javax.swing.*;
import javax.swing.text.*;
import java.awt.event.*;
import javax.swing.event.MouseInputListener;
import java.util.Hashtable;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
```

```
public class StandardEditActions
{
    private static Hashtable g_hashClasses = new
Hashtable();     // map of classes to
StandardEditActions (to encourage re-use).

    private Hashtable m_actions = null;          //
these could be shared between components...
    private JPopupMenu m_menu = null;


/*******************************************************
******************
     * Nested class. Handles the binding between the
MouseAdapter and the text component
     *
     */
    class StandardEditActionComponent extends
MouseAdapter
    {
        private JTextComponent m_textcomp;          //
the text component this applies to

        private
StandardEditActionComponent(JTextComponent component)
        {
            m_textcomp = component;
            m_textcomp.addMouseListener(this);         //
for the popup menu
        }

        public void mouseClicked(MouseEvent me)
```

```
            {
                if (me.isPopupTrigger() ||
SwingUtilities.isRightMouseButton(me))
                {
                    JPopupMenu menu = buildPopup();

                    String str =
m_textcomp.getSelectedText();
                    boolean bSel = (str != null &&
str.length() > 0);
                    boolean bEditable =
m_textcomp.isEditable();

getActionByName(DefaultEditorKit.cutAction).setEnabled(
bSel && bEditable);

getActionByName(DefaultEditorKit.copyAction).setEnabled
(bSel);

getActionByName(DefaultEditorKit.pasteAction).setEnable
d(bEditable);

                    menu.show(m_textcomp, me.getX(),
me.getY());
                }
            }
        }


/*****************************************************
****************
```

```
      * Adds standard cut/copy/paste menu to the
specified component.
      */
     public static void addStandardMenu(JTextComponent
component)
     {
         Class c = component.getClass();
         StandardEditActions sea =
(StandardEditActions)g_hashClasses.get(c);
         if (sea == null)
         {
             sea = new StandardEditActions();
             g_hashClasses.put(c, sea);
         }
         sea.addComponent(component);
     }

     private void addComponent(JTextComponent component)
     {
         createActionTable(component);
         new StandardEditActionComponent(component);
     }

     private StandardEditActions()
     {
     }

     // construct the popup menu
     JPopupMenu buildPopup()
     {
         if (m_menu == null)
         {
             JPopupMenu menu = new VPopupMenu();
```

```
            // According to the docs we should really work on copies of the default
            // actions. Otherwise changing the name might impact other bits of Swing.
            Action actionCut = renameAction(DefaultEditorKit.cutAction, "Cut");
            menu.add(actionCut);

            Action actionCopy = renameAction(DefaultEditorKit.copyAction, "Copy");
            menu.add(actionCopy);

            Action actionPaste = renameAction(DefaultEditorKit.pasteAction, "Paste");
            menu.add(actionPaste);

            menu.addSeparator();

menu.add(renameAction(DefaultEditorKit.selectAllAction, "Select All"));
            m_menu = menu;
        }
        return m_menu;
    }


/*****************************************************************************************************
     * Gets the action table for a text components and puts it into a hash
     * table, so we can find actions easily
     */
    private void createActionTable(JTextComponent
textcomp)
    {
        m_actions = new Hashtable();
        Action[] actionsArray = textcomp.getActions();
        for (int i = 0; i < actionsArray.length; i++)
        {
            Action a = actionsArray[i];
            m_actions.put(a.getValue(Action.NAME), a);
        }
    }


/*******************************************************
*******************
     * Retrives the Action corresponding to the
specified name
     */
    private Action getActionByName(String name)
    {
        return (Action)(m_actions.get(name));
    }



/*******************************************************
*******************
     * Wraps a supplied action with an ActionDelegate,
and sets the menu name
     * to be the supplied text.
     */
```

```
        private Action renameAction(String name, String
    menuname)
        {
            Action action = getActionByName(name);  // get
    the original object in our list

            // wrap the object. If we've already wrapped it
    (shouldn't happen), then
            // don't wrap it again.
            if (!(action instanceof ActionDelegator))
            {
                ActionDelegator action2 = new
    ActionDelegator(action);  // wrap it in a delegate
                m_actions.put(name, action2);       //
    update our list
                action = action2;
            }

            // set the name of the menu text. NB: we don't
    update the name in the hash
            // table - otherwise the DefaultEditorKit
    constants won't work to find the name
            action.putValue(Action.NAME, menuname);
            return action;
        }


    }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/StringTok
enizer2.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

/**
 * Slot-in replacement for StringTokenizer, but which
allows the remaining text on a line to be extracted
with getRemainder()
 * @author $Author: markc $
 * @version $Revision: 1.2 $
 * $Date: 2002/04/15 08:53:01 $
 */

import java.util.Enumeration;


public class StringTokenizer2 implements Enumeration
{
    private String m_str;
    private String m_Delims;    // delimiters
    private int m_index = 0;    // char to start at

    public StringTokenizer2(String str, String delims)
    {
        m_str = str;
        m_Delims = delims;
        skipChars(false);
    }
```

```
/*******************************************************
*****************
     * Skip characters.
     * @param skipChars true-> skip characters, false-
>skip delimiters
     */
    final private void skipChars(boolean skipChars)
    {
        while (m_index < m_str.length())
        {
            char ch = m_str.charAt(m_index);
            if (isDelimiter(ch) == skipChars)   // is
delimiter
                break;
            m_index++;
        }
    }


    // returns true if the character is a delimiter
    final private boolean isDelimiter(char ch)
    {
        return (m_Delims.indexOf(ch) >= 0);
    }

    public boolean hasMoreTokens()
    {
        return (m_index < m_str.length());    // a non-
space is next.
    }

    public String nextToken()
```

```
        {
            int nSt = m_index;
            skipChars(true);
            String token = m_str.substring(nSt, m_index);
            skipChars(false);  // skip delimiters at the
    end
            return token;
        }


    /*****************************************************************
    ****************
         * Returns the remainder of the string upto the
    current point
         * the whole point of this method !
         */
        public String getRemainder()
        {
            return m_str.substring(m_index);
        }

        // Enumeration implementation
        public boolean hasMoreElements()
        {
            return hasMoreTokens();
        }

        public Object nextElement()
        {
            return nextToken();
        }
}
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/VPopupMen
u.java
//+++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

/**
This component extends JPopupMenu and adds a method to
display the menu inside the screen, even if the mouse
pointer is near the edge of the screen.
Use directly in place of JPopupMenu
<br>
Class created by Cris Sinnott
<br>
Source was available at :
http://www.egroups.com/list/advanced-
java/md1875700976.html
*/


import javax.swing.*;
import java.awt.*;

public class VPopupMenu extends JPopupMenu
{

    /**Displays the popUpMenu at a specified position*/
    public void show(Component invoker, int x, int y)
    {
        Point p = getPopupMenuOrigin(invoker, x, y);
        super.show(invoker, p.x, p.y);
    }
    /**Figures out the sizes needed to calculate the
menu position*/
    protected Point getPopupMenuOrigin(Component
invoker, int x, int y)
    {
        Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
        Dimension pmSize = this.getSize();
        // For the first time the menu is popped up
        // the size has not yet been initialised
        if (pmSize.width == 0)
        {
            pmSize = this.getPreferredSize();
        }

        Point absp = new Point(x,y);
        SwingUtilities.convertPointToScreen(absp,
invoker);
        int aleft = absp.x + pmSize.width;
        int abottom = absp.y + pmSize.height;

        if(aleft > screenSize.width)
            x -= aleft - screenSize.width;

        screenSize.height -= 50;            // fudge
factor to give some allowance for the windows taskbar
        if(abottom > screenSize.height)
            y -= abottom - screenSize.height;

        return new Point(x,y);
    }

}
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/VTextFiel
d.java
//++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

import javax.swing.*;
import javax.swing.text.*;
import java.awt.event.MouseEvent;
import java.awt.event.ActionEvent;
import javax.swing.event.MouseInputListener;
import java.util.Hashtable;

/**
 * A JTextComponent with standard cut/copy/paste menus
 * @author $Author: markc $
 * @version $Revision: 1.1 $
 */
// $Date: 2002/03/19 17:23:45 $

public class VTextField extends JTextField
{
    public VTextField()
    {
        super();
        StandardEditActions.addStandardMenu(this);
    }

    public VTextField(int n)
    {
        super(n);
        StandardEditActions.addStandardMenu(this);
        }

        public VTextField(String str)
        {
            super(str);
            StandardEditActions.addStandardMenu(this);
        }
    }
```

```
//Source file:
./chat/ChatExtension/src/com/fs/chatextension/WholeNumb
erField.java
//+++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
package com.fs.chatextension;

/*****************************************************
*******************
* JTextField derived component that ONLY allows
integers to be entered
* Unashamedly nicked from the JFC tutorial.
* $Date: 2002/02/19 17:17:20 $
* $Author: markc $
* $Revision: 1.1 $
*****************************************************
******************/

import javax.swing.*;
import javax.swing.text.*;

import java.awt.Toolkit;
import java.text.NumberFormat;
import java.text.ParseException;
import java.util.Locale;

public class WholeNumberField extends JTextField {
    private Toolkit toolkit;
    private NumberFormat integerFormatter;

    public WholeNumberField(int value, int columns) {
        super(columns);
        toolkit = Toolkit.getDefaultToolkit();
```

```
            integerFormatter =
    NumberFormat.getNumberInstance(Locale.US);
            integerFormatter.setParseIntegerOnly(true);
            setValue(value);
        }

        public int getValue() {
            int retVal = 0;
            try {
                retVal =
    integerFormatter.parse(getText()).intValue();
            } catch (ParseException e) {
                // This should never happen because
    insertString allows
                // only properly formatted data to get in
    the field.
                toolkit.beep();
            }
            return retVal;
        }

        public void setValue(int value) {
            setText(integerFormatter.format(value));
        }

        protected Document createDefaultModel() {
            return new WholeNumberDocument();
        }

        protected class WholeNumberDocument extends
    PlainDocument {
            public void insertString(int offs,
                                     String str,
                                          AttributeSet a)
                                 throws BadLocationException {
                            char[] source = str.toCharArray();
                            char[] result = new char[source.length];
                            int j = 0;

                            for (int i = 0; i < result.length; i++) {
                                if (Character.isDigit(source[i]))
                                    result[j++] = source[i];
                                else {
                                    toolkit.beep();
                                    System.err.println("insertString: "
                + source[i]);
                                }
                            }
                            super.insertString(offs, new String(result,
                0, j), a);
                        }
                    }
                }
```

What is Claimed is:

1. A method for using a web site in association with a chat room to create a topic room, comprising:

storing content on the web site;

passing a message that comprises a reference to the content to the associated chat room;

allowing users of the associated chat room to view the message; and

allowing the users to view or play the content by using the reference to the content within the message.

2. The method defined in claim 1 further comprising allowing objects to listen to messages and take scripted actions in response.

3. The method defined in claim 1 further comprising enabling objects to send messages to the associated chat room describing their actions.

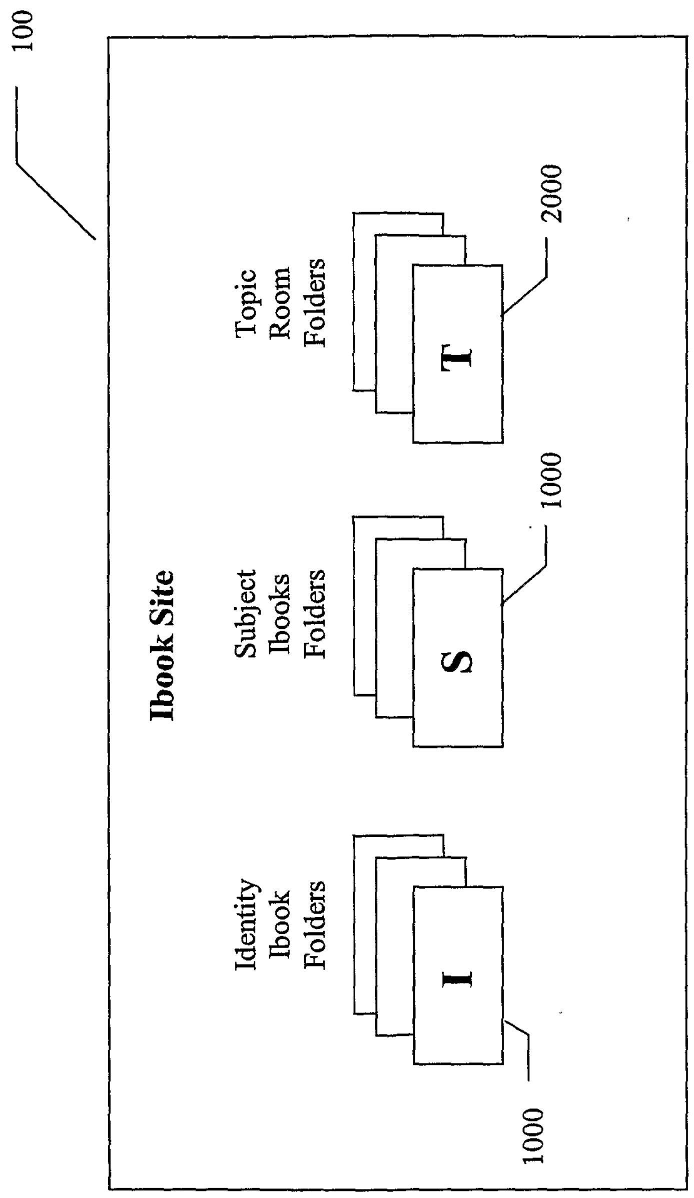

100
Ibook Site
Identity Ibook Folders
I
1000
Subject Ibooks Folders
S
1000
Topic Room Folders
T
2000

Fig 1

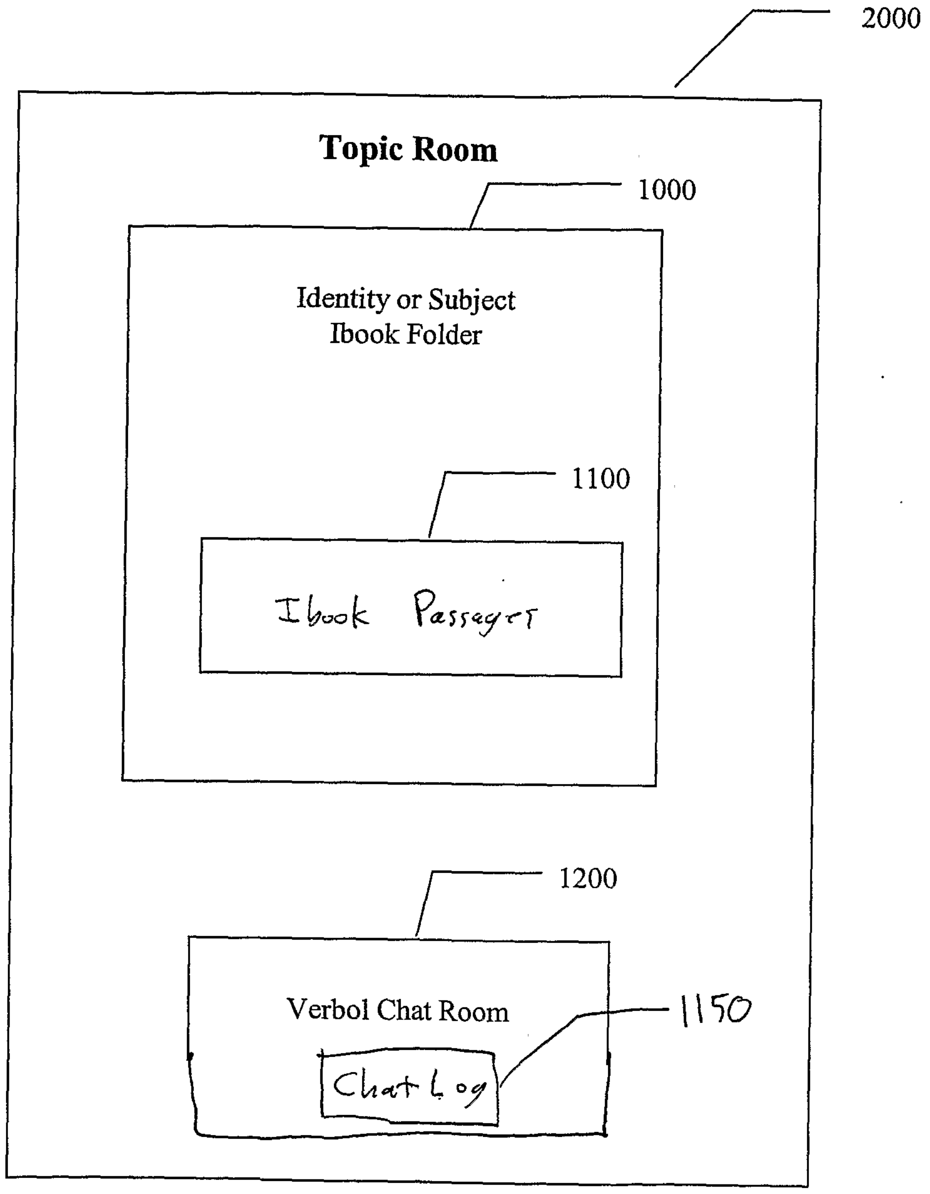

2000
Topic Room
1000
Identity or Subject
Ibook Folder
1100
Ibook Passages
1200
Verbol Chat Room
1150
Chat Log

Fig 2

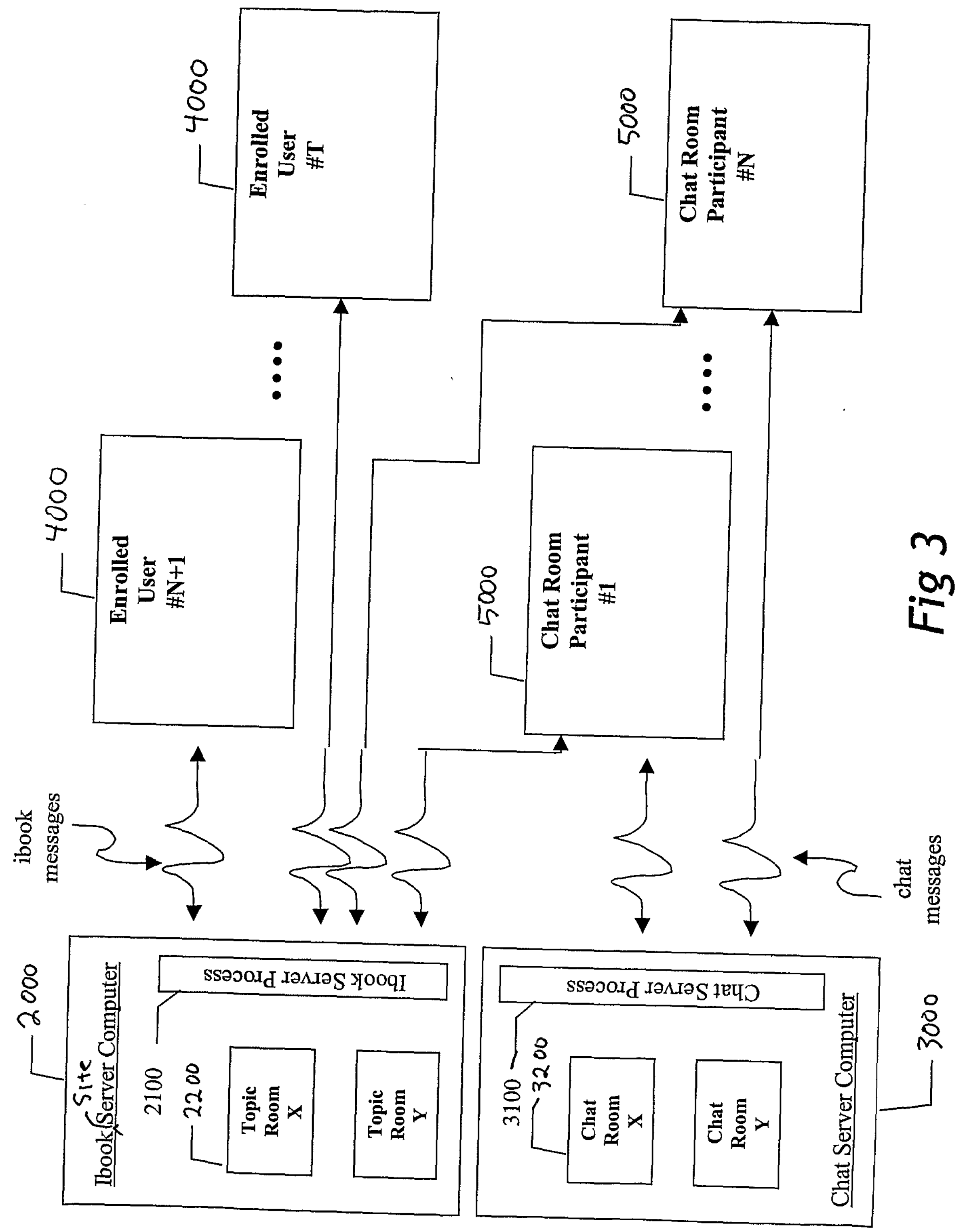

Enrolled User #N+1
4000
Enrolled User #T
4000
Chat Room Participant #1
5000
Chat Room Participant #N
5000
ibook messages
chat messages
Ibook Site Server Computer
2000
Ibook Server Process
2100
2200
Topic Room X
Topic Room Y
Chat Server Computer
3000
Chat Server Process
3100
3200
Chat Room X
Chat Room Y

Fig 3

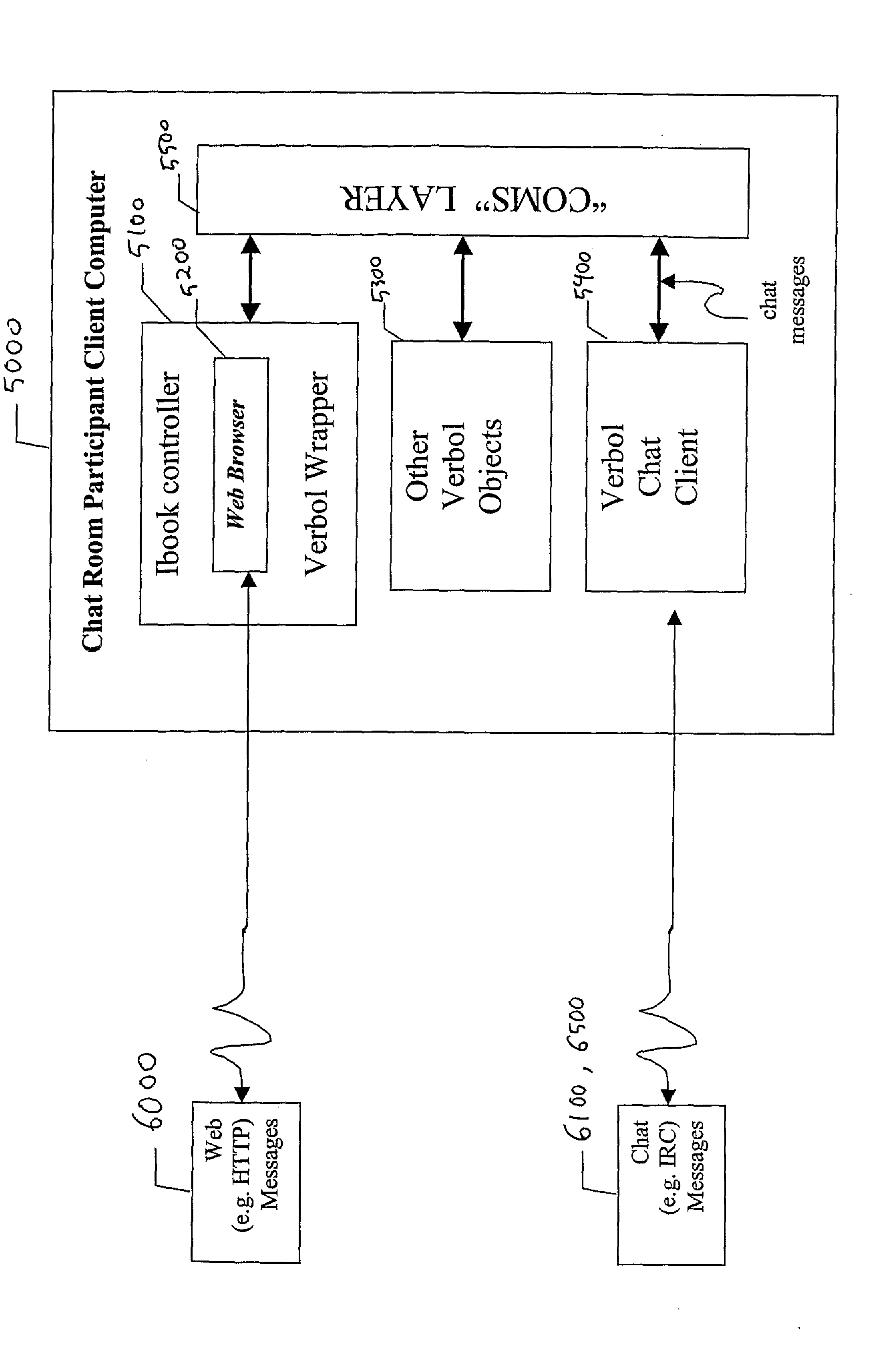

5000
Chat Room Participant Client Computer
5100
5200
Ibook controller
Web Browser
Verbol Wrapper
5500
"COMS" LAYER
5300
Other Verbol Objects
5400
Verbol Chat Client
chat messages
6000
Web (e.g. HTTP) Messages
6100, 6500
Chat (e.g. IRC) Messages

Fig 4

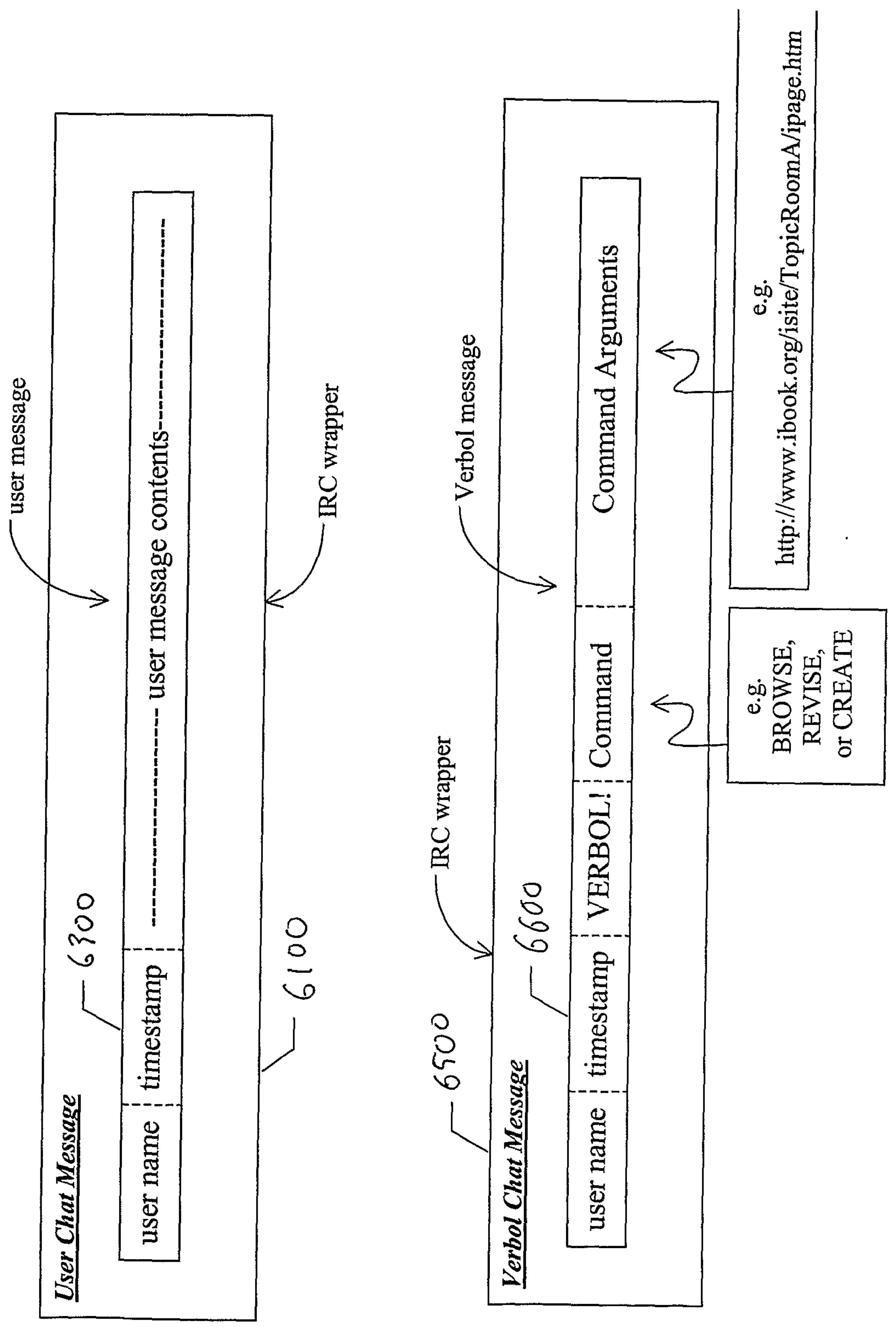
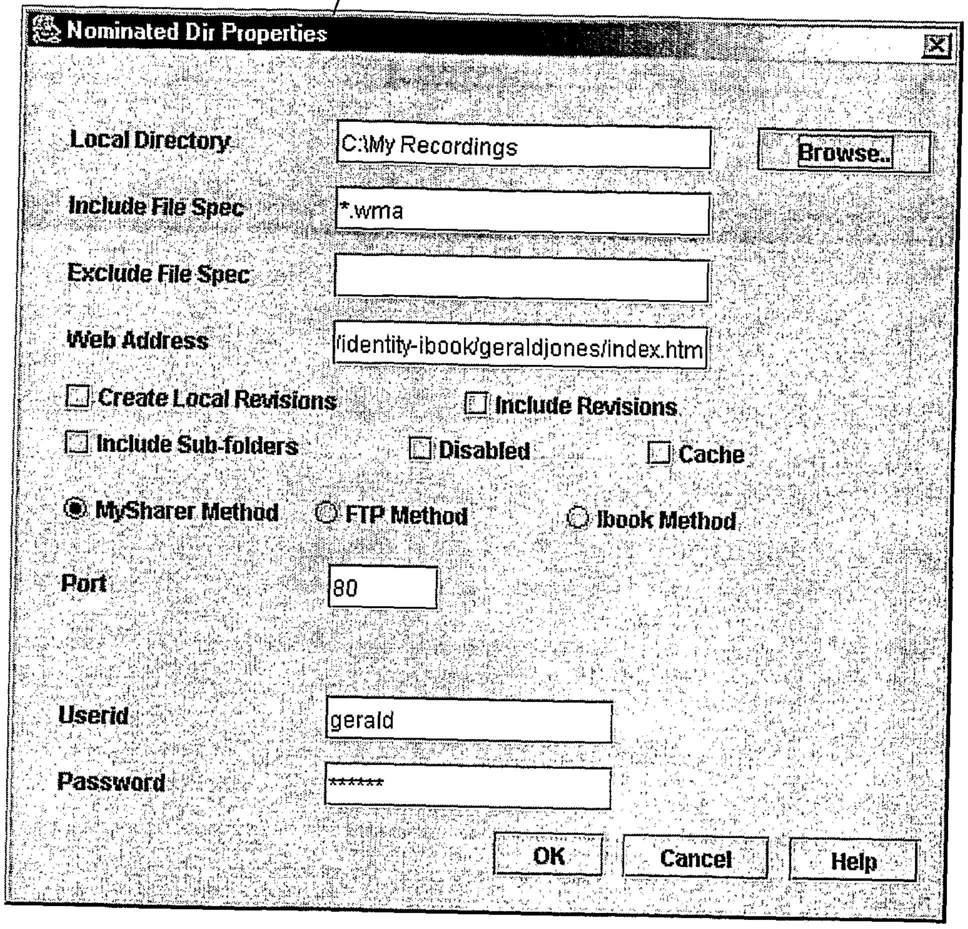

User Chat Message
user name
timestamp
------------------ user message contents------------------
user message
IRC wrapper
6300
6100
Verbol Chat Message
user name
timestamp
VERBOL!
Command
Command Arguments
Verbol message
IRC wrapper
6500
6600
e.g. BROWSE, REVISE, or CREATE
e.g. http://www.ibook.org/isite/TopicRoomA/ipage.htm

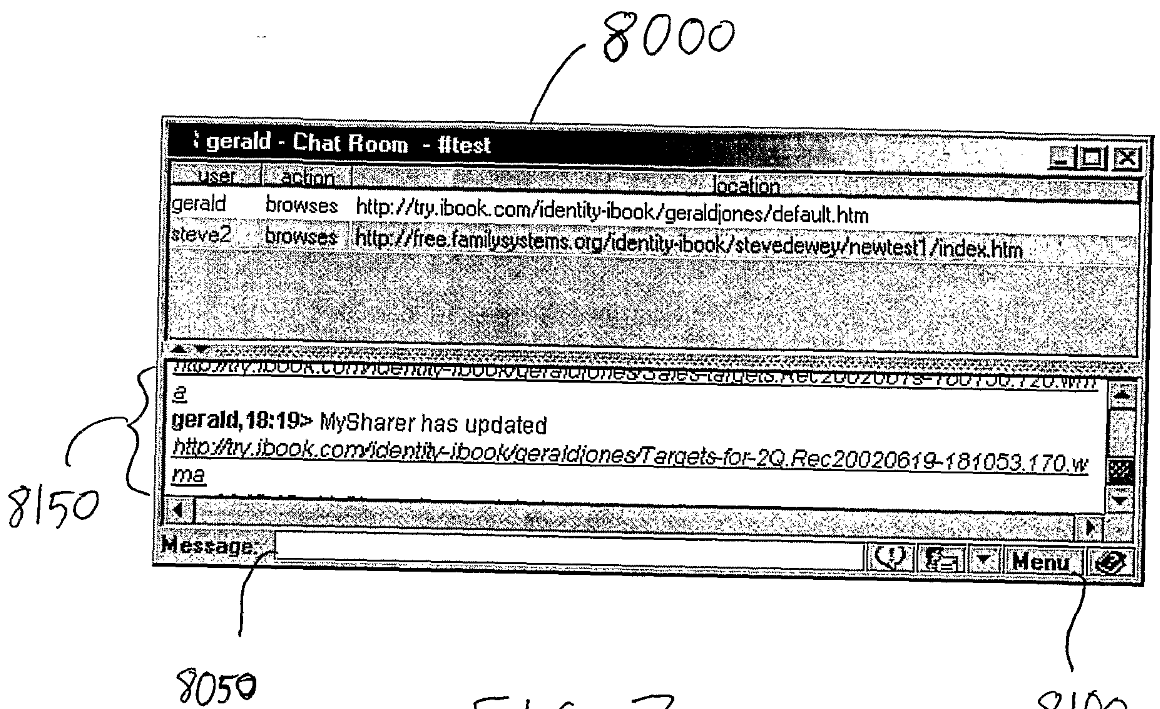
Fig 5

7000



Local Directory: C:\My Recordings
Include File Spec: *.wma
Exclude File Spec:
Web Address: /identity-ibook/geraldjones/index.htm
☐ Create Local Revisions ☐ Include Revisions
☐ Include Sub-folders ☐ Disabled ☐ Cache
[✓ MySharer Method] FTP Method Ibook Method
Port: 80
Userid: gerald
Password: ******

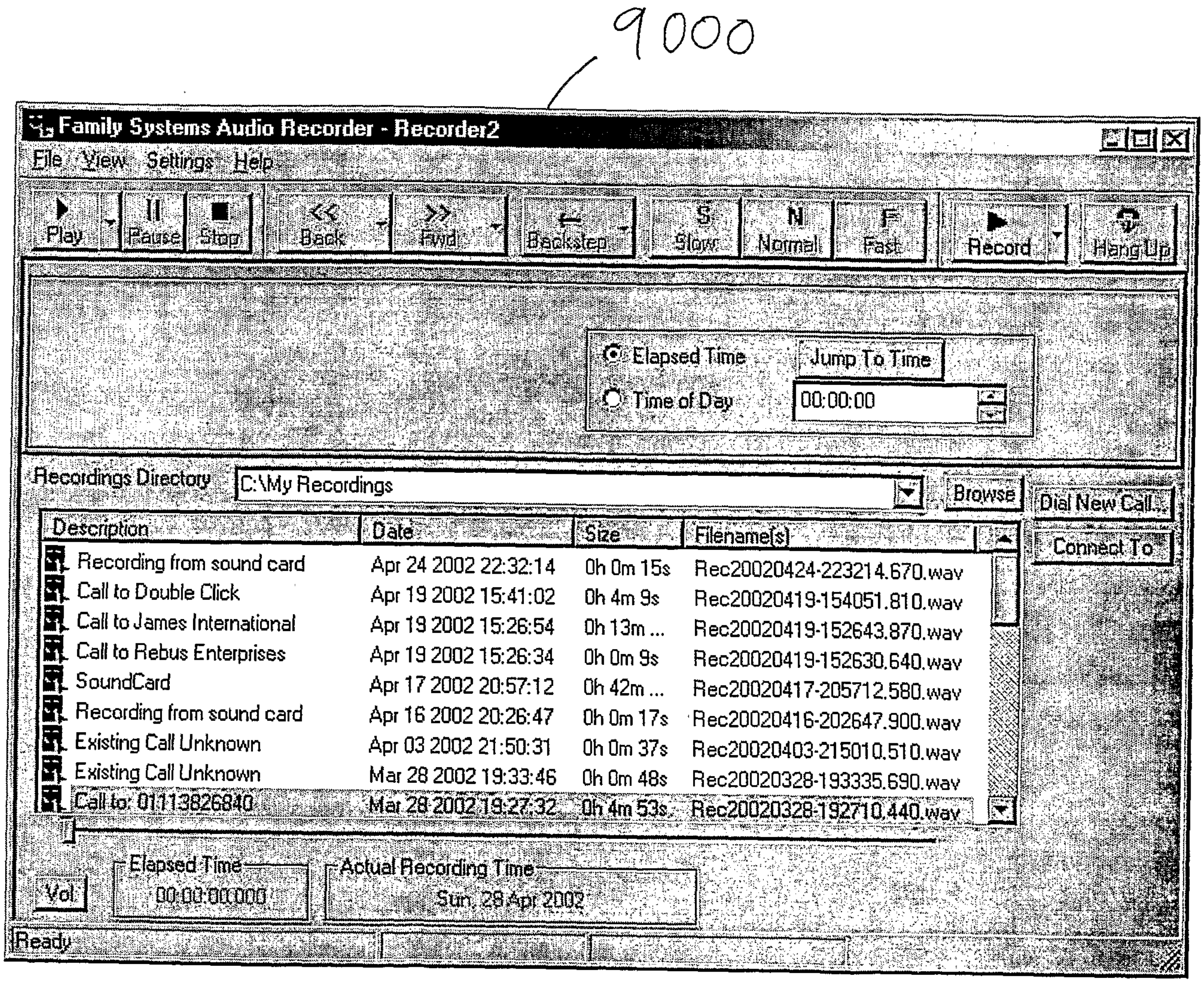
FIG. 6



user	action	location
gerald	browses	http://try.ibook.com/identity-ibook/geraldjones/default.htm
steve2	browses	http://free.familysystems.org/identity-ibook/stevedewey/newtest1/index.htm

a
gerald,18:19> MySharer has updated
http://try.ibook.com/identity-ibook/geraldjones/Targets-for-2Q.Rec20020619-181053.170.wma

FIG. 7



[✓ Elapsed Time] Time of Day

00:00:00

Recordings Directory C:\My Recordings

Description	Date	Size	Filename(s)
Recording from sound card	Apr 24 2002 22:32:14	0h 0m 15s	Rec20020424-223214.670.wav
Call to Double Click	Apr 19 2002 15:41:02	0h 4m 9s	Rec20020419-154051.810.wav
Call to James International	Apr 19 2002 15:26:54	0h 13m ...	Rec20020419-152643.870.wav
Call to Rebus Enterprises	Apr 19 2002 15:26:34	0h 0m 9s	Rec20020419-152630.640.wav
SoundCard	Apr 17 2002 20:57:12	0h 42m ...	Rec20020417-205712.580.wav
Recording from sound card	Apr 16 2002 20:26:47	0h 0m 17s	Rec20020416-202647.900.wav
Existing Call Unknown	Apr 03 2002 21:50:31	0h 0m 37s	Rec20020403-215010.510.wav
Existing Call Unknown	Mar 28 2002 19:33:46	0h 0m 48s	Rec20020328-193335.690.wav
Call to: 01113826840	Mar 28 2002 19:27:32	0h 4m 53s	Rec20020328-192710.440.wav

Elapsed Time: 00:00:00.000

Actual Recording Time: Sun, 28 Apr 2002

Ready

FIG. 8