INTERNET RADIO AND BROADCAST METHOD

TECHNICAL FIELD

This invention relates to Internet media data streams and the like, and more particularly to a copyright-compliant audio/video/radio broadcast system over the Internet where each individual user is able to set his or her preferences regarding works played so as to influence the frequency such works are broadcast to the user.

BACKGROUND ART

The rise of the Internet has provided many different channels through which media can be presented to users. RealNetworks' RealMedia, Apple QuickTime, and Windows Media all provide players through which live or previously-recorded data streams can be displayed, played back, or broadcast to the individual user. Both audio and video are generally available through these programs and provide a higher and more attractive degree of interactivity with the Internet.

Regular radio broadcasts are based upon a central individual or station broadcasting songs, or other audio information, electromagnetically. Different radio stations are separated by their different carrier frequencies. Amplitude modulation (AM) and frequency modulation (FM) provide two means by which radio broadcast can be effected by a transmitter to a receiver. If an individual wants to affect the songs that are played by the radio station, he or she may write, call, fax, e-mail, or otherwise transmit their preferences to the radio station.

However, one person's preferred music may not be as appreciated by another individual. Music can be very personal, often affecting a person at an emotional level. When the radio station broadcasts a song or other audio signal, all receivers tuned to the carrier frequency pick up the broadcast and either enjoy or suffer the broadcast equally.

It would be much more advantageous to allow each individual to influence, their own set of song playlists. Currently, this is not achievable by wireless broadcast means. However, unique data stream addressing available through Internet data processing might provide means by which an Internet radio could be advantageously affected. Other Internet broadcasting processes are known, but generally follow the known radio station format of broadcasting a single song, or data stream, to all users tuned to the station or channel. In compliance with the Digital Millennium Copyright Act (DMCA), such a radio would have to comply with statutory regulations regarding the broadcast of songs and would generally have to avoid the role of an "on-demand" system, as this might be in violation of statutory regulation.

The following patents may have some bearing on the art relevant to the present invention:

U.S. PATENT NUMBER	INVENTOR	DATE OF ISSUE
6,052,717	Reynolds et al.	April 18, 2000
6,038,591	Wolfe et al.	March 14, 2000
6,031,797	Van Ryzin et al.	February 29, 2000
6,026,439	Chowdhury et al.	February 15, 2000
5,987,525	Roberts et al.	November 16, 1999
5,945,988	Williams et al.	August 31, 1999
5,930,768	Hooban	July 27, 1999
5,864,868	Contois	January 26, 1999
5,819,160	Foladare et al.	October 6, 1998

U.S. PATENT NUMBER	INVENTOR	DATE OF ISSUE
5,809,246	Goldman	September 15, 1998
5,790,423	Lau et al.	August 4, 1998
5,758,257	Herz et al.	May 26, 1998
5,740,134	Peterson	April 14, 1998
5,726,909	Krikorian	March 10, 1998
5,721,827	Logan et al.	February 24, 1998
5,661,787	Pocock	August 26, 1997
5,616,876	Cluts	April 1, 1997
5,592,511	Schoen et al.	January 7, 1997
5,539,635	Larson, Jr.	July 23, 1996

DISCLOSURE OF INVENTION

The present invention provides a copyright-compliant, broad-based, individually-tailored Internet media broadcast system and method. The present invention provides means by which users may individually rate or indicate music, music videos, or other recorded media that they enjoy hearing from a vast musical or other database. Additionally, such users may also indicate the exclusion of music/media that is to their distaste. In so doing, the user interaction is limited to that decision-making role that is necessary for the user to establish his or her preferences. The Internet radio of the present invention and its method take care of the rest, providing the end user a media or radio channel tailored to his or her own musical tastes. In this way, the present invention can be said to "microcast," or "narrowcast" the content of personalized songlists to individual listening stations or users. As the broadcast uses Internet protocol, each data packet of each data stream has its own individual address, namely, the end-user's data stream player. As the present invention is scalable, thousands, even tens or hundreds of thousands of listeners can be handled by the present invention. With the advance of data-transmission technology, tens or hundreds of millions of users may be served by, or given access to, a system incorporating the present invention, including the delivery of user-preferred data streams by wireless communication links.

Mention is made herein of the present invention with respect to music broadcast to provide a personalized Internet, or data stream, radio. Note should be taken that use of the term "radio," "music," and the like includes any recorded datastream content, including music videos and the like.

At the core of the present invention is the playlist generator. It is the generated songlist that is associated with the user's account and indicates to the system which song is to be played next. Once a song has been selected, it is then streamed as data out to the individual's computer (uniquely identified by Internet protocol). As the central server of the system can handle a large number of users at any one time, it becomes possible to serve each user with his or her own individual data stream. In this case, the data stream comprises audio and/or video information and serves to establish a situation similar to each user having his or her own individual radio station that he or she programs. The list can be created in advance and stored, or generated, in real time when needed. Collaborative filtering techniques may be used in constructing the playlist.

Other applications for the present method may also exist when similar circumstances are present where a large database of information is available that is subject to individual preferences. In a broad sense, the present invention provides means by which individual subsets of an all-encompassing data space may be defined, modified, and preserved, subject to a variety of influences and allowing some serendipitous, or random, events to occur.

BRIEF DESCRIPTION OF DRAWINGS

Figure 1 is a schematic view of the system architecture used to achieve one embodiment of the present invention.

Figure 2 is a screen shot showing a computer desktop with the audio player and user homepage for the present invention.

Figure 3 is a screen shot showing a computer desktop with the video player and user homepage for the present invention.

BRIEF DESCRIPTION OF APPENDICES

The following appendices are incorporated herein by this reference thereto.

Appendix 1 is an excerpted text listing of a playlist generated in conformance with the present invention.

Appendix 2 is a source code listing for one embodiment of the present invention.

MODE(S) FOR CARRYING OUT THE INVENTION

The detailed description set forth below in connection with the appended drawings is intended as a description of presently-preferred embodiments of the invention and is not intended to represent the only forms in which the present invention may be constructed and/or utilized. The description sets forth the functions and the sequence of steps for constructing and operating the invention in connection with the illustrated embodiments. However, it is to be understood that the same or equivalent functions and sequences may be accomplished by different embodiments that are also intended to be encompassed within the spirit and scope of the invention.

This patent application is related to United States Provisional Patent Application Serial Number 60/164,846 filed November 10, 1999 for Internet Radio and Broadcast Method, which application is incorporated herein by this reference thereto.

As mentioned above, use of the term "radio," "music," and the like includes any recorded datastream content, including music, videos, recorded sports events and concerts, and the like.

In Figure 1, the general structure of the present system is shown where the LAUNCHcast Player provides user feedback and indication of song preference through Java Servlets and JavaScript code. In one embodiment, a Windows Media Player may provide the interface allowing the audio and/or video broadcast to take place at the user's computer. Other media players now known or developed in the future may also suffice and operate to good advantage. Mentioned use of the Windows Media Player system is to be considered as indicating any appropriately functioning media player. Song or video information is available through both the player and the accompanying data window.

Referring now to Figure 1, the architecture and system structure of the Internet radio and broadcast method of the present invention is shown in schematic form. The system **100** is generally focused upon the player **102**. The player **102** is the component that the user sees and is ultimately the arbiter of the media datastream service provided by the present invention. As shown in Figure 1, the player **102** has a song information section **104**, a rating tool **106**, and a player **108**. For this last component, the player **108** is indicated as being a Windows Media player. However, other media players can also be used to good advantage in order to achieve the present invention.

Through its components, the player **102** is linked or associated to a number of other sources of information and programs, including Java or other servlets. The present invention, when implemented in software, may be so implemented using Java-family of computer program languages. A servlet is Java programming that runs as a part of a

network service, such as an HTTP server, in response to requests from clients. In this case, the client can be considered to be the player **102** while the HTTP server can be the servers for the database **160** and the media content library **180**.

At a center of the present invention is the player **108**. The player **108** allows the content to be broadcast to the individual user and serves as means by which the user can enjoy such content. In addition to being linked to the media database **180**, the player **108** is also in communication with a media gateway servlet **120** and a playlist generator servlet **122**. As discussed in more detail below, these two servlets provide the player the ability to play streaming media in conformance with the present invention.

The rating tool **106** is coupled to the database **160** via a rating acceptor servlet **130** and a ratings cache servlet **132**. As indicated in Figure 1, the rating acceptor servlet **130** and ratings cache servlet **132** are also in communication with one another, as set forth in more detail below.

The song information component **104** of the player **102** may provide links to other information available through the database **160** or otherwise. For example, the song information tool **104** may provide links to other user pages **140**, a station manager **142**, provided home pages of various artists **144**, as well as links to album pages **146** of such artists or otherwise. Additionally, a central homepage **148** may be present that allows travel or linking to any or all of available pages or services.

Note should be taken that the database **160** is not necessarily the home for the media library **180**. In fact, according to present technology, it may be more advantageous to provide some means by which high-speed access can be provided to the media library **180**. By separating the database **160** from the media library **180** faster and better service may be provided to users so they may enjoy the content of datastream better. Certain infrastructures may allow for offsite residence of the media contained in the media library **180**. Pointers or other indicators to such information in an indexed or other form can thereby provide the link necessary to deliver the preferred or indicated content by the user from the media library **180** to that same user.

As shown in Figure 1, the database **160** may hold a variety of types of information, including: user data **162**, playlists **164**, and song data **166**. Such information is stored by the database **160** and updated by the servlets as set forth in the present invention, including the user code set forth in Appendix 2.

In Figure 2, the player, or playback, window **102** is shown and is highly interactive with several embedded hyperlinks. In the upper right-hand corner of the playback window **102**, the indication of "asjordan" is made. By clicking on this link, more information about the current station may be given and/or the ability to change such station. The user's page **140** may be activated and shown upon clicking the username link. In the right center of the playback window, a "RATE IT" window indicator that is the rating tool **106** is given, allowing the individual to rate the current "SONG", the "ARTIST" performing the current song, and/or an "ALBUM" containing the song. Below the "RATE IT" indicator, hyperlinks to "RECENT SONGS", "BUY", and "STATION MANAGER" are present allowing the user to travel to those destinations and either learn more information, purchase or review purchasing information about the current album being played, as well as access the station manager for the present invention.

Below the song information window **104**, icons are given for Play/Pause, Skip This Song, Skip This Song and Never Play It Again ("Delete"), and a Volume control. The question mark ("?") shown below the "Song Information area" window is a hyperlink to a Help file for the playback window **102** and the Internet Radio system of the present invention. These icons are also shown in the other playback window Figures, such as that for the video playback user interface/client **102** shown in Figure 3.

Figures 2 and 3 show a desktop display of the system **100** in action from the user's point of view. A tool tip may be given when the cursor hovers over the song title. The same may be similarly true for the artist and the album

currently playing. Note should be taken that just as the song rating indicator is highlighted and active in the middle right section of the playback window, the song title is highlighted in the upper portion of the playback window.

Additionally, the left and center middle portion of the playback window provides information regarding fans who have strong positive feelings about the present song, artist, and/or album, as well as an average rating for all users or some subset of users on the system.

Figures 2 and 3 show small balloons on the right-hand side of the central dark area across from the "Fans." These balloons may have a letter "W" inside of them to indicate another listener is currently online and can be engaged via the instant messaging ("whisper") function. Figures 2 and 3 also show graphic information that may be used for advertising or other hyperlinks. In generating the playlist of the present invention, the user can be informed as to why a particular song was picked.

For other links and presentation of information in the player **102**, a tool tip may be presented when the cursor hovers over an area. A tool tip is a small window providing succinct information about the item under the cursor when the cursor hovers over that item.

When the system **100** is updating and obtaining a new data stream from the system for the user, a display may be given to the user to indicate ongoing activity of the playback system. Such visual activity in the form of animation assures the listener/viewer that the short span of silence, or "dead air," following a song is only temporary and that a new song will soon play. Generally, in order to promote interactivity and to take advantage of the new media that the Internet provides, the windows shown in the Figures 2 and 3 contain ample internal hyperlinks that lead to web pages providing information regarding music, artists **144**, and/or their works **146**, web pages regarding other users of the system (as DJs or otherwise) **140**, and/or web pages regarding the user's control of the system (preferences, etc.) **142**.

The default paradigm for the user interface/player **102** is to allow the user the greatest degree of freedom in expressing preferences and in obtaining that preference information regarding music artists, and their publications/albums. In this way, the user's experience is enhanced as he or she hears more of the music he or she likes. Access to purchasing web sites is also made available where users may purchase artists' works.

In implementing the present invention in software, the accompanying source code (Appendix 2) may be used to achieve the present invention. Such code is subject to copyright protection and is owned by LAUNCH Media, Inc. of Santa Monica, California.

The generation of a proper playlist combining available user ratings and a media database forms an important part of the present invention. One such playlist as generated by the present invention is shown in Appendix 1 and is an excerpted form for purposes of explanation. Entries in the playlist have been removed so that the playlist may better serve the explanatory purposes herein without undue length or the sacrifice of sufficient detail.

Playlist generation occurs when a user launches his client player **102**. A Windows Media or other player **108** is embedded in the user's client player **102**. The player **108** opens a call to the playlist generator servlet **122** as executed by the PlaylistGeneratorServlet routine (Appendix 2, page 158). The expected output from this HTTP call is an ASX playlist file, which in the present invention is list of pointers to a script that reads the actual playlist data object from the database **160**.

The playlist generator servlet **122** parses the particular parameters for this ASX playlist as follows:

Object: GeneratorParameters;

userID: (required) the user for whom the playlist is generated;

djID: (default is userID) the user whose profile will be used to generate the playlist;

moodID: (default is none) a mood which is a subset of a profile may be indicated and used to alter the preferences

in the playlist and under which to listen (optional); and

bandwidth: (default is 28.8k, if not read from the user's preferences in the database) the bit rate at which the user wishes to listen.

The database **160** with the playlist database **164** is checked for an existing playlist by PlaylistStatus (Appendix 2, page 192). If a playlist already exists, it can be used it if all the following are met (and PlaylistStatus.isStale() returns false):

all of the parameters (userID, djID, etc) match;

there are more than 8 songs left;

the newRatingsCount (counter of new personalization data since last refresh) is less than 15; and

the playlist is less than a week old.

If all these conditions are met, the dates for the last time the user listened to an ad, news bit, and tip may be reset and the playlist may be resaved. The ASX file is written out and media player begins to execute by making requests to the media gateway **120** to play music.

If the old playlist cannot be used, a new one is created with the playlist generator via PlaylistGenerator.create().

The first step is to retrieve the user's preferences via PlaylistGenerator.getOptions(). In response the following options are returned:

unratedQuota: how much new (not rated) music they want hear in their playlist. The options here are 90, 80, 70, 50, 40, 30, and 20 percent. The default is 50 percent.

explicit lyrics: Does this user want us to play music with explicit lyrics? True or false.

bandwidth: if the bandwidth is not already specified in the generator parameters, it is read from stored data. Currently, bandwidth options include 28.8, 56, and T1/LAN. The default is 28.8 if a valid setting of "none" is found in the database.

A list of all the possible songs available for play (via PlaylistGenerator.gatherMedia()) as well as some other data about those songs is obtained. This is generally done using multiple threads running at the same time for better performance. The list of songs is held in hashtable (as via the Population subroutine (Appendix 2, page 198)).

The database **160** is first called to load a history of all the songs played for the user in the last 30 days. This is stored in the database as a long string, formatted as: "<Date>=<songID>,<Date>=<songID>, . . . " For performance reasons, reading one string from the database is faster than reading potentially several thousand rows individually from the database. Dates older than 30 days are ignored and the last time a song was played overwrites previous plays of a song. Each time a song is played via the media gateway **120**, this string is appended.

After the history loading is complete, a random integer is picked from 1 to 10. If the value is 1, the date and songID string is recreated and rewritten to the database. This cleans up the string by removal of songs that were played more than 30 days ago as well as duplicate entries for the same songID.

The history loads as a thread, and another database call is made to get the user's, or DJ's, list of subscribed DJs, genres, and radio stations (via PlaylistGenerator.getSubscriptions()) for the specific mood requested. The result of this call is three lists called DJs, genres, and stations.

Once the subscriptions are available, the ratings are obtained via GetRatings. This is also done in a thread. The song hashtable, another hashtable that contains Artist and Album ratings (ItemsProfile), the DJ, and the list of subscribed DJs are all passed to the GetRatings method routine.

A retrieval list of users whose ratings are to be retrieved is compiled using the subscribed DJs and the DJ requesting the playlist. A request is made to the ratings cache to retrieve all these ratings via RatingsCache.getRatings().

When the playlist generator has all the ratings, it is ready to assemble them into categorized data structures, based on the properties of each rating. It iterates through all the ratings and stores them in the following manner. If the ID of the user is the DJ and the rating is 0 (an 'X' in the end-user interface), the song is added to song hashtable (via Population) as an "Excluded" type, meaning that song should never be played. The rating is also added to the average rating for songs by that artist. If the rating is not 0, the song information cache is immediately checked via SongInfoCache.get() for data about this song. If the data does not exist in the cache, it is a song that was rated, but is not available for play (as possibly not encoded), and the song is immediately marked as an "Excluded" song.

If all of the above tests pass, the song is added to the song hashtable with a type of "Explicit". The rating for the song is included in the calculation of this DJ's average rating of songs by the artist.

Each song that is rated by subscribed DJs is added to the song hashtable. The subscribed DJ's rating for the song is included in the calculation of the subscribed DJs' average rating for this song.

For albums, the ratings profile is obtained from the item rating profiles. If a ratings profile for an album does not yet exist, then the data regarding the album is retrieved and a ratings profile is created.

If the rater is the user requesting the playlist, the rating for this item is set to the user's rating. However, if the rater is a subscribed DJ, the rating is added to the DJ's average for this album.

For artists, the rating procedure is the same as for albums, except any ratings made for the artists listed as "Various Artists", "Soundtrack", or "Original Soundtrack" are discarded or ignored in the relevant calculations.

The top 1000 most popular songs (via PlaylistGenerator.getPopular()) in the bandwidth type specified may be added to the song candidate hashtable. This popular list is maintained in the song information cache. Before each song is added to the song hashtable, inspection is made to see if the song is already in the candidate hashtable (perhaps put there by another query). If so, inspection is made to make sure that the song is not of type "Excluded", or the song is discarded. If the song is added to the song hashtable, it is added under the type "Unrated".

A maximum of 5000 songs are picked randomly (via PlaylistGenerator.getRandom()). Initially, a count is made of the number of songs contained in each and all of the genres a user has selected (via SongInfoCache.countInGenres()). Songs may be in multiple genres. The number of songs is then divided by the total number of songs in the song information cache. If the result is less than 5%, songs are picked directly from a list of songs only in those genres. Otherwise, songs can be picked randomly from all available songs. This calculation may be performed to avoid the situation where a user has selected a small number of genres and picking songs randomly will return only a few songs that are available or allowable for play when considering their genres.

In order to select songs only from selected genres, a determination is made of the total number of songs to pick (via totalToPick) from the lesser of 5000 and the total number of songs in the selected genres. For each genre, a copy of the list of songs in that genre is obtained from the song information cache (via SongInfoCache.getInGenre()). The number of songs to pick from each genre is determined from the following formula: songs to pick = totalToPick * (number of songs in this genre / total number of songs in the selected genres).

The determined number of songs is picked and attempts are made to add the songs to the song hashtable with a type of "Unrated". A song is not added if it is already in the hashtable.

In order to select from all songs, a song is randomly selected 5000 times. Each time, attempts are made to add the song if it is not already there as picked, as described above. Once the process finishes adding random songs, all the ratings for the songs are retrieved as are all the dates of when the songs were played for the user. The explicit, implicit, and unrated lists built in the last step are taken and ordered in descending order by score, or rating, using a quicksort or other algorithm.

The number of songs to pick from each list is determined. For example, if the size of a playlist is 50 songs, the following may occur. If the user is listening to his own station, the following formula may be used: if the user's list of explicit and implicit songs is smaller than 100 songs, 90% of the songs must be picked from the unrated list to avoid playing the user's rated songs too much. The user's unrated quota may, then, be set to 90. Otherwise, an unrated quota may be used from the user's stored options.

Under some circumstances the maximum number of songs available from the explicit and implicit song lists is calculated as follows:

maximumRated = playlistSize * (100 - unratedQuota) * 0.01.

The maximum number of songs available from the explicit list may be calculated as:

MaximumExplicit = number of songs in the explicit list * .20.

A number of songs to pick from the explicitly-rated list may then be:

explicitToPick = playlistSize * (100 - unrated quota) * 0.01 * (number of songs in the explicit list / sum of explicit and implicit songs) * 3);

From this the number of implicit songs is simply:

implicitToPick = maxiumumRated – explicitToPick.

Confirmation can be made to ensure that more explicit songs have not been picked than indicated by maximumExplicit and that no more implicit songs have been picked than those that are in the implicit list. The number of unrated songs is then: playlistSize - (explicitToPick - implicitToPick)

If the user is listening to a station other than his own and the number of songs in the explicit and implicit list total greater than 200, then the following calculations are made:

explicitToPick = Minimum(playlistSize * .50, 20% of explicit songs); and

implicitToPick = Minimum(playlistSize, # of implicit songs) - explicitToPick

If, for some reason, a sufficient and/or playlistSize number of songs is not obtained from this calculation, a third of the songs is picked from each of explicit, implicit and unrated songs with a check to ensure that not more than 20% of the songs on the rated and unrated lists are picked. As a fallback measure if none of the methods above used to calculate the number of songs to pick worked, the songs are selected as a third of the playlistSize from each list, making sure not to pick more than 20% of the rated and unrated lists.

A list of albums and artists from and by which songs have been played for this user in the last 3 hours is copied or otherwise made available to the process set forth herein and the songs for this playlist are picked via PlaylistGenerator.pickSongs(). A list of all the picks needed is made (via PickList). For example, if there is a playlist of 50 songs, the list may contain 10 entries for explicit songs, 20 for implicit songs, and 20 for unrated songs.

While there are still songs to pick, iteration is made through the following cycle:

a. randomly pick a song list type (explicit, implicit, unrated) with a probability based on the proportion of songs to come from each list;

b. pick a random song index from that list (which has already been sorted in descending order of score), based on the following formula (via SongGroup.pickRandom()):

sizeOfList = the number of songs in this list;

random = a randomly-chosen number between 0 and (sizeOfList - 1) + 0.01; and

index of song to pick = ((rand ^ 7) / sizeOfList - 1 ^ 7) * (sizeOfList - 1)).

This formula allows songs to be picked somewhat randomly, while guaranteeing a high probability that the song picked will come from highest scored. The higher the ranking of the song in the score matrix, the higher the probability

it will be picked. This algorithm scales well for any size of list because it is rank-based, not just score based.

The song at that index is removed from the list. If for some reason a valid song is not obtained (possibly the song list already exhausted), another song is added to the list of types to pick of this type.

Once a song is picked, its album and artist information are obtained.

If the artist is not a "Various Artist" and the sum of the number of songs played by this artist and already picked for this playlist by this artist is greater than or equal to 3, this song cannot be played under the RIAA (Recording Industry Associates of America) and/or DMCA (Digital Millennium Copyright Act) rules. Other rules may also be implemented in the present invention to accommodate statutory and other rights and/or restrictions.

The song is marked as "rejected" and another song is added to the list of songs to pick from the same list the rejected song was picked from. The same test is performed for albums, with the maximum played, for example, being 2. If the song was picked successfully and was within legal or other boundaries, the number of songs picked from this album and by this artist is incremented. The song is added to the final list of songs for the playlist and the order in which the song was picked for the playlist is marked, or noted.

If, for some reason, a playlistSize number of songs is not obtained, the existing playlist is deleted and popular songs are added to the song hashtable, and the song lists are re-sorted and re-picked ignoring the user's genres selections.

The picking of news clips is done simply by picking a specific number of unique news items that are in the specified bandwidth format. A list of available news clips is stored in the song information cache. Ads may be picked in the same way as news clips are picked. However, a difference may be present in the different number of ads to pick. Tips may also be picked in the same manner as news clips, with a different number of tips to pick.

The order of the songs may be randomly shuffled in the playlist and the playlist may be serialized and saved to the database. Finally, the ASX file may be returned to the player **108**.

Every 5 minutes, the player **102/108** "pings" the Playlist Generator **122**. If the playlist is stale or has 8 songs or less left in it, the playlist generator regenerates the playlist and replaces the one previously saved in the database.

As an additional enhancement to the present invention, playlists from commercial and other radio stations throughout the United States, and elsewhere, are made available so that playlists may be affected by such radio stations and by popularity of particular musical works.

In achieving the Internet radio of the present invention, a rating acceptor **130** in the form of the RatingWidgetServlet routine (Appendix 2, page 222) takes HTTP requests to rate and gets ratings for songs, albums, and artists. When a rating is saved, it written to the ratings database and if the user who rated the item is designated as being in the ratings cache, the rating change is added to the queue of ratings updates.

Once every minute, the ratings updates are sent to all the ratings caches that have registered their IP address in the database. Every hour, the list of ratings caches are retrieved from the database. Every ten minutes, the list of users in the cache are retrieved from the database.

The song information cache is implemented through the SongInfoCache routine (Appendix 2, page 265) and may be a large in-memory cache of relatively static data that is used in playlist generation. It may include a list and hashtable of all songs which includes identifying numbers, media formats available, average rating, artist and album information, explicit lyrics mark, genres the song is in, and radio stations that play the song. Also, other information may be included in the song information cache, including: a hashtable of artist information;, a hashtable of album information; a list and hashtable of all ads including identifying numbers and media formats available; a list and hashtable of all news clips including identifying numbers and media formats available; a list and hashtable of all audio tips including identifying numbers and media formats available; a lists of the 1000 most popular songs in each media format; lists of all songs in

each genre; and a cache of frequently-accessed ratings profiles. This last cache is seen in the RatingsCache 132 routine (Appendix 2, page 211). The song information cache is completely rebuilt once a day from the database.

The ratings cache caches the entire ratings profile for the top 100 users who are known to be accessed frequently. The ratings cache is implemented through the RatingsCache routine (Appendix 2, page 211). On startup, the ratings cache registers its IP address in the database to subscribe to ratings updates. These users are typically DJs (users with broadcasted or subscribed ratings) that have many subscribers, or users who simply use LAUNCHcast frequently. Each ratings cache recalculates the most frequently-accessed users and writes it to the database every 8 hours. At that time, the entire cache is discarded and reread from the database to erase any lingering corruption. Each ratings cache checks the database every 10 minutes for changes in the list of users to be cached and updates the ratings cache as appropriate.

Note should be taken that many of the parameters set forth herein are discretionary and advisory. Consequently, those properly and legitimately implementing the present invention may alter such parameters, such as when events occur and event timing as above, according to system operation preferences.

For each user who is not in the ratings cache, their ID is appended to a list of users whose profiles need to be retrieved from the database 160. Users who have been added to the cache recently have their profiles added to the list of ratings to be returned to the PlaylistGenerator 122 routine (Appendix 2, page 158). All non-cached users' ratings are retrieved from the database 160, are appended to the list of ratings, and are returned to the PlaylistGenerator 122. The album and artist ratings are retrieved in a separate query from the song ratings. Each runs in its own thread in parallel for optimal performance.

The media gateway 120 is a Java servlet that brokers the relationship between the end user's (Windows Media) Player 108, the database 106, and media library, or Windows Media Server, 180 and logs all media access. The MediaGatewayServlet routine (Appendix 2, page 112) performs this function. Because the client's Windows Media Player playlist (.sax file) does not contain any information about the actual songs or ads in the user's playlist, the media gateway 120 contains the logic described below to redirect the user's player to the correct media address on the media library 180.

For security reasons, the media gateway 120 may check to see that the client 102 is accessing it from the Windows Media Player client 108 (and not a web browser or other application). If not, it may redirect the user to an error media file. The media gateway 120 then pulls the user's ID off the query string and retrieves that user's playlist object from the database 160. The gateway 120 inspects timestamps in the user's playlist object that indicate when the user last heard an ad, tip, song or other media item and determines if it is time to insert an ad, tip, or news item in the datastream, or just play the next song.

If the user has not heard an ad, for example, for a pre-defined period of time, the media gateway 120 resets an ad timestamp and retrieves an ad path from the user's ad playlist and passes that MMS (Microsoft Media Server) redirect instruction/address to the end user's Windows Media client 108. If no ad is available, the process continues and plays the next song in the user's playlist. If it is not time to play an ad, the timestamp is checked to see if it is time to play a tip. The process then follows the same logic, above, for ads to retrieve and play a tip, instead of an ad. If it is not time to play an ad or tip, the timestamp is checked to see if it is time to play a news item. The process then follows the same logic as for ads to retrieve and play a news item.

If it is not time to play an ad, tip, news item, or other stream (the usual case), the media gateway 120 retrieves the path of the next song in the playlist and returns that address via an MMS redirect to the client's Windows Media Player 108. In all cases, the mediaID of the ad, tip, or song played is logged in the database 160 under that user's ID. This logging information is used to display what the user is listening to on the user's station page and under the "Who's

Listening" page. These pages may be associated with the central home page **148** in a manner similar to that of the user pages **140** as history data in the playlist generator, and in calculating a Top 100 chart for the most popular songs and/or streams.

While there may be some preference for an "on-demand" service such that individuals may pick their own radio playlists, the element of randomness and pleasant surprise is inherent in the present invention. Additionally, statutory requirements prevent users from turning the Internet into their own home stereo system. "On-demand" service is generally prevented by statute and may be a violation of copyright. Consequently, any statutory regulations, such as the Digital Millennium Copyright Act (DMCA), and other limitations can be programmed automatically into the present invention. In so doing, the present invention complies with all applicable law and delivers to the user a musical experience generally aligned with his or her preferences.

Many users often listen to music while doing programming or the like. Such music can now be delivered over the Internet via the user's very own radio station through the present invention. Additionally, users may select other individuals or DJs, to influence their musical playlist just as the user does. The DJ, online or otherwise, becomes an additional factor in influencing the user's preferences and playlist. Some individuals may act as real DJs, serving to provide content to an audience of subscribers through the Internet. Programs of special interest may also be developed and subscribed to by listeners using the present invention. Through the heavily hyperlinked (but easily understandable) interface set forth in the Figures and described above, a user may establish musical (or other data stream) preferences. In establishing such preferences, the music played to the listener is tailored to that listener and provides an enhanced musical experience on an individual basis.

While the present invention has been described with reference to a preferred embodiment or to particular embodiments, it will be understood that various changes and additional variations may be made and equivalents may be substituted for elements thereof without departing from the scope of the invention or the inventive concept thereof. In addition, many modifications may be made to adapt a particular situation or material to the teachings of the invention without departing from the essential scope thereof. Therefore, it is intended that the invention not be limited to particular embodiments disclosed herein for carrying it out, but that the invention includes all embodiments falling within the scope of the appended claims.

INDUSTRIAL APPLICABILITY

It is an object of the present invention to provide individualized data stream programming according to an individual's preference.

It is yet another object of the present invention to provide an Internet-based radio or music playing system that is biased according to each user's preferences.

It is yet another object of the present invention to provide a means by which song playlists may be generated for such an Internet radio.

It is an object of the present invention to provide copyright-compliant media streams for Internet and other networked systems broadcast.

These and other objects, advantages, and the industrial utility of the present invention will be apparent from a review of the accompanying specification and drawings.

Playlist status for userID 6474126:
newRatingsCount: 0
moodID: 0
djID: 6474126
songsRemaining: 50
mediaType: 212
generating because forceRefresh is on
regenerating playlist with parameters: userID=6474126, bandwidth=28.8k, moodID=0, djID=6474126<PRE>
start of createPlaylist

0.0 lap time, 0.0 total
---

starting gathering threads at

0.0 lap time, 0.0 total

GetLastPlayed loaded 618 dates
getSubscriptions done

0.063 lap time, 0.063 total

All threads started

0.0 lap time, 0.063 total
---

getPopular done

0.047 lap time, 0.11 total
---

getRandom done (picked 5000 songs)

1.281 lap time, 1.391 total
---

genres for mood 0:64, 44, 46, 48, 50, 45, 47, 49, 51, 63, 67, 1, 0, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24, 68, 69, 73, 74, 75, 76, 77, 78, 79, 80,
gatherMedia done

0.0 lap time, 1.391 total
---

scores calculated

0.156 lap time, 1.547 total
---

recently played albums and artists marked

0.0 lap time, 1.547 total

Of 6749 songs, these are the reasons for exclusion: 546 were already excluded, 349 were not encoded, 34 were played in the last 3 hours, 0 had explicit lyrics, 0 were not in mediaType 212, 1292 were not in their genres, 482 had an implicit rating of 0.
There are 4046 songs available for play
ordering...

0.0 lap time, 1.547 total

---

finished sorting vectors at

0.11 lap time, 1.657 total

Available: explicit songs: 388.0, implicit songs: 2334.0, unrated songs: 1324.0
Ratio: 20
Picking: explicit songs: 17, implicit songs: 23, unrated songs: 10, method = Unrated Ratio
start of pickSongs

0.0 lap time, 1.657 total

---

end of pickSongs

0.0 lap time, 1.657 total

---

picked news

0.0 lap time, 1.657 total

---

picked ads

0.015 lap time, 1.672 total

---

picked tips

0.0 lap time, 1.672 total

---

playlist has 50 songs
shuffling playlist...
end of createPlaylist

0.0 lap time, 1.672 total

---

starting to save playlist

0.016 lap time, 1.688 total

---

done saving playlist

0.031 lap time, 1.719 total

</PRE>
<PRE>
Playlist 0 for userID 6474126 (djID 6474126) in mood 0 with mediaType 212, pickCounts: explicit to pick: 17, implicit to pick: 23, unrated to pick: 10 has 50 songs:
37409 146690 1022473 1364151 Emitt Rhodes Listen, Listen: The Best Of Emitt Rhodes You're A Very Lovely Woman - The Merry-Go- Round)
37718 43307 1016600 385563 Madonna Erotica Erotica
45680 43305 1016600 385517 Madonna The Immaculate Collection Cherish
40237 98477 1025497 900407 Squeeze The Piccadilly Collection * Loving You Tonight

21825 132410 1027798 1212736 U2 The Best Of 1980-1990 [Limited] New Year's Day
37268 137097 1028125 1259519 Various Artists Made On Earth Untitled - Total Eclipse
8405 41860 1015576 372519 The Lightning Seeds Sense Sense
31547 91874 1015450 839523 Jackie Leven Forbidden Songs Of The Dying West Birds Leave Shadows
42209 100072 1028125 1407544 Various Artists Assemblage Vol. 1 Taksu - Lights in a Fat City
39401 105661 1005547 956525 Paula Cole This Fire * Tiger
52454 85650 1024526 778897 Carly Simon Clouds In My Coffee 1965-1995 [Box] Stuff That Dreams Are Made Of, The
53486 51128 1021142 458446 Pink Floyd Ummagumma Narrow Way Part 1, The - David Gilmour
17982 58282 1025027 526886 Social Distortion Prison Bound Backstreet Girl
22578 14393 1000398 123761 Bryan Adams So Far So Good Summer Of '69
6947 130669 1009757 1193855 Fun Lovin' Criminals 100% Columbian * Big Night Out
39632 113337 1028125 1011924 Various Artists Pure Moods Crockett's Theme - Jan Hammer
30674 93944 1028256 857682 The Verve Pipe Villains * Cattle
28189 61860 1026856 559756 They Might Be Giants They Might Be Giants Toddler Hiway
16788 23890 1005543 212417 Jude Cole Start The Car Right There Now
37247 137097 1028125 1259512 Various Artists Made On Earth Portnawack - Typhoon
28606 64190 1030389 578647 Vanilla Fudge Rock & Roll Windmills Of Your Mind, The - (original mix)
6299 118154 1005865 1062093 Cornershop When I Was Born For The 7th Time * Brimful Of Asha
29369 74082 1025801 673069 Sting Fields Of Gold: The Best Of Sting 1984-1994 They Dance Alone (Cueca Solo)
23334 148558 1026856 1386237 They Might Be Giants Miscellaneous T Kiss Me, Son Of God - (alternate version)
53363 50728 1021142 454344 Pink Floyd A Saucerful Of Secrets Let There Be More Light
50557 50901 1020983 455893 Tom Petty Into The Great Wide Open All Or Nothin'
42791 142342 1025039 1327416 Soft Cell Non-Stop Ecstatic Dancing Insecure Me
30719 95006 1021869 867248 R.E.M. New Adventures In Hi-Fi Wake-Up Bomb, The - (live)
42923 148836 1015285 1388605 Ben Lee Breathing Tornados * Cigarettes Will Kill You
39860 123837 1018539 1122003 Morcheeba Big Calm Friction
30644 93944 1028256 857672 The Verve Pipe Villains * Drive You Mild
31529 91874 1015450 839517 Jackie Leven Forbidden Songs Of The Dying West Working Alone/A Blessing
39320 92012 1028514 841099 Loudon Wainwright III Grown Man Human Cannonball
22344 143220 1000012 1331978 10,000 Maniacs The Earth Pressed Flat * [4/20] Hidden In My Heart
26698 47344 1018869 423656 Peter Murphy Should The World Fail To Fall Apart God Sends
21660 130952 1021402 1196259 Portishead PNYC * Strangers
26686 47344 1018869 423652 Peter Murphy Should The World Fail To Fall Apart Light Pours Out Of Me, The
39137 87489 1023065 798733 David Lee Roth The Best Lil' Ain't Enough, A
7646 145523 1030217 1352144 Buddy Holly 20th Century Masters:... [4/20] Maybe Baby
44144 25421 1006149 227025 Crosby, Stills & Nash CSN [Box] Southern Cross
21999 135883 1038686 1242702 The Hope Blister Smile's OK... Is Jesus Your Pal
39644 113337 1028125 1011928 Various Artists Pure Moods Theme From "Twin Peaks - Fire Walk With Me" - Angelo Badalamenti
50515 50895 1020983 455822 Tom Petty Full Moon Fever Face In The Crowd, A
40510 117098 1018623 1049778 Morrissey Maladjusted He Cried
31805 87741 1013181 801582 Jars Of Clay Jars Of Clay Like A Child
29384 74082 1025801 673074 Sting Fields Of Gold: The Best Of Sting 1984-1994 We'll Be Together - (previously unreleased version)
25621 36886 1012859 328927 INXS X Disappear
28039 60022 1025830 544499 The Stone Roses Second Coming Love Spreads
26269 41495 1015374 369132 Lemonheads Come On Feel The Lemonheads Into Your Arms
52466 85650 1024526 778868 Carly Simon Clouds In My Coffee 1965-1995 [Box] Better Not Tell Her

2 songs are by the artist Jackie Leven (1015450)
1 songs are by the artist Bryan Adams (1000398)
1 songs are by the artist Paula Cole (1005547)
1 songs are by the artist Soft Cell (1025039)
1 songs are by the artist Portishead (1021402)
2 songs are by the artist They Might Be Giants (1026856)
1 songs are by the artist Crosby, Stills & Nash (1006149)
1 songs are by the artist Vanilla Fudge (1030389)
1 songs are by the artist Jude Cole (1005543)
2 songs are by the artist Carly Simon (1024526)
2 songs are by the artist Peter Murphy (1018869)
1 songs are by the artist Social Distortion (1025027)

2 songs are by the artist The Verve Pipe (1028256)
2 songs are by the artist Tom Petty (1020983)
1 songs are by the artist The Stone Roses (1025830)
1 songs are by the artist Fun Lovin' Criminals (1009757)
1 songs are by the artist Morcheeba (1018539)
1 songs are by the artist R.E.M. (1021869)
1 songs are by the artist Jars Of Clay (1013181)
1 songs are by the artist Emitt Rhodes (1022473)
5 songs are by the artist Various Artists (1028125)
2 songs are by the artist Sting (1025801)
1 songs are by the artist Squeeze (1025497)
1 songs are by the artist Morrissey (1018623)
1 songs are by the artist David Lee Roth (1023065)
2 songs are by the artist Madonna (1016600)
1 songs are by the artist Ben Lee (1015285)
2 songs are by the artist Pink Floyd (1021142)
1 songs are by the artist INXS (1012859)
1 songs are by the artist Loudon Wainwright III (1028514)
1 songs are by the artist U2 (1027798)
1 songs are by the artist Lemonheads (1015374)
1 songs are by the artist The Lightning Seeds (1015576)
1 songs are by the artist Buddy Holly (1030217)
1 songs are by the artist 10,000 Maniacs (1000012)
1 songs are by the artist Cornershop (1005865)
1 songs are by the artist The Hope Blister (1038686)

1 songs are from the album The Best Of 1980-1990 [Limited] (132410)
1 songs are from the album Into The Great Wide Open (50901)
1 songs are from the album Full Moon Fever (50895)
1 songs are from the album Miscellaneous T (148558)
1 songs are from the album Come On Feel The Lemonheads (41495)
1 songs are from the album When I Was Born For The 7th Time * (118154)
1 songs are from the album 20th Century Masters:... [4/20] (145523)
1 songs are from the album Assemblage Vol. 1 (100072)
1 songs are from the album Erotica (43307)
1 songs are from the album The Immaculate Collection (43305)
2 songs are from the album Should The World Fail To Fall Apart (47344)
1 songs are from the album 100% Columbian * (130669)
1 songs are from the album Jars Of Clay (87741)
1 songs are from the album CSN [Box] (25421)
1 songs are from the album New Adventures In Hi-Fi (95006)
2 songs are from the album Forbidden Songs Of The Dying West (91874)
1 songs are from the album Breathing Tornados * (148836)
1 songs are from the album PNYC * (130952)
1 songs are from the album Rock & Roll (64190)
1 songs are from the album Start The Car (23890)
1 songs are from the album So Far So Good (14393)
2 songs are from the album Fields Of Gold: The Best Of Sting 1984-1994 (74082)
1 songs are from the album They Might Be Giants (61860)
1 songs are from the album Sense (41860)
2 songs are from the album Made On Earth (137097)
1 songs are from the album Maladjusted (117098)
1 songs are from the album Smile's OK... (135883)
1 songs are from the album Listen, Listen: The Best Of Emitt Rhodes (146690)
1 songs are from the album Non-Stop Ecstatic Dancing (142342)
1 songs are from the album Second Coming (60022)
1 songs are from the album A Saucerful Of Secrets (50728)
1 songs are from the album The Best (87489)
1 songs are from the album Ummagumma (51128)
1 songs are from the album X (36886)
2 songs are from the album Pure Moods (113337)
1 songs are from the album This Fire * (105661)

2 songs are from the album Villains * (93944)
1 songs are from the album Big Calm (123837)
1 songs are from the album Prison Bound (58282)
1 songs are from the album The Earth Pressed Flat * [4/20] (143220)
2 songs are from the album Clouds In My Coffee 1965-1995 [Box] (85650)
1 songs are from the album The Piccadilly Collection * (98477)
1 songs are from the album Grown Man (92012)

21 songs (42.0%) are from the random query
6 songs (12.0%) are from the pop query
6 songs (12.0%) are from the djs query
17 songs (34.0%) are from the rated query

3 songs (6.0%) originated from djAlb
11 songs (22.0%) originated from random
3 songs (6.0%) originated from djs
6 songs (12.0%) originated from s avg
3 songs (6.0%) originated from artist
7 songs (14.000000000000002%) originated from album
17 songs (34.0%) originated from rated

Percentile 0% - 20%: 40 (80%)
Percentile 20% - 40%: 2 (4%)
Percentile 40% - 60%: 2 (4%)
Percentile 60% - 80%: 4 (8%)
Percentile 80% - 100%: 2 (4%)

<P>
Item Ratings

Artist "The Cure" (1006316) user=0(Not Set) djs=50/1=(Not calculated) songAverage=0/0=(Not calculated) songAvgScore=0.0
Artist "Liz Phair" (1020993) user=30 djs=70/1=70 songAverage=0/0=(Not calculated) songAvgScore=0.0
Artist "Freaky Chakra" (1009573) user=0(Not Set) djs=0/0=(Not calculated) songAverage=0/1=0 songAvgScore=39.0
Artist "Duncan Sheik" (1024246) user=0(Not Set) djs=0/0=(Not calculated) songAverage=80/1=80 songAvgScore=59.0
Artist "Tom Petty" (1020983) user=73 djs=20/1=20 songAverage=554/8=(Not calculated) songAvgScore=0.0
Album "Great Divide" (94571) user=0(Not Set) djs=70/1=(Not calculated) songAverage=0/0=(Not calculated) songAvgScore=0.0
Album "Devil Without A Cause *" (127191) user=20 djs=0/0=(Not calculated) songAverage=0/0=(Not calculated) songAvgScore=0.0

«entries omitted».

Artist "Iron City Houserockers" (1012883) user=0(Not Set) djs=0/0=(Not calculated) songAverage=0/2=0 songAvgScore=26.0
Album "Superunknown" (58747) user=0(Not Set) djs=70/1=70 songAverage=0/0=(Not calculated) songAvgScore=0.0
Artist "To Rococo Rot" (1032453) user=0 djs=0/0=(Not calculated) songAverage=0/0=(Not calculated) songAvgScore=0.0
Album "(Not available)" (132141) user=0(Not Set) djs=80/1=(Not calculated) songAverage=0/0=(Not calculated) songAvgScore=0.0
Album "Buckcherry" (143554) user=0(Not Set) djs=50/1=50 songAverage=0/0=(Not calculated) songAvgScore=0.0
Artist "Jamie Blake" (1030814) user=0(Not Set) djs=60/1=60 songAverage=0/0=(Not calculated) songAvgScore=0.0
Album "(Not available)" (45683) user=90 djs=0/0=(Not calculated) songAverage=0/0=(Not calculated) songAvgScore=0.0
Album "(Not available)" (45676) user=90 djs=0/0=(Not calculated) songAverage=0/0=(Not calculated) songAvgScore=0.0
Artist "INXS" (1012859) user=0(Not Set) djs=70/1=70 songAverage=69/2=35 songAvgScore=43.5

Artist "Kenny Wayne Shepherd" (1024272) user=0(Not Set) djs=0/0=(Not calculated) songAverage=0/1=(Not calculated) songAvgScore=0.0
Album "The Ghost Of Tom Joad" (89708) user=0(Not Set) djs=0/1=0 songAverage=0/0=(Not calculated) songAvgScore=0.0
Artist "(Not available)" (1001434) user=0(Not Set) djs=10/1=(Not calculated) songAverage=0/0=(Not calculated) songAvgScore=0.0
Explicitly Rated Songs

#	songID	query comm	origin albumID	status artisID	ord artist	score title	lastP. album	bds	impl.	rating(t) djs	netP.
1	372519	rated 52/0	rated 46/0	P 41860	5 1015576	79 The Lightning Seeds	100/30	0/0	49 Sense	70/49 (1) Sense	52/0 (14, 77,)
2	385517	rated 52/0	rated 49/0	P 43305	9 1016600	79 Madonna	100/30	0/0 Cherish	49 The Immaculate Collection	70/49 (1)	52/0 (14, 28, 77,)
3	673074	rated 52/0	rated 51/0	P 74082	14 1025801	79 Sting	100/30 We'll Be Together - (previously unreleased version) Fields Of Gold: The Best Of Sting 1984-1994	0/0	49	70/49 (1) (14, 77,)	52/0
4	673069	rated 52/0	rated 44/0	P 74082	18 1025801	79 Sting	100/30 They Dance Alone (Cueca Solo) Fields Of Gold: The Best Of Sting 1984-1994	0/0	49	70/49 (1) (14, 77,)	52/0
5	123761	rated 52/0	rated 48/0	P 14393	22 1000398	79 Bryan Adams	100/30 Summer Of '69	0/0	49 So Far So Good	70/49 (1)	52/0 (13, 14, 23, 77,)
6	1388605	rated 52/0	rated 55/0	P 148836	19 1015285	79 Ben Lee	100/30 Cigarettes Will Kill You	0/0	49 Breathing Tornados *	70/49 (1) (14, 77,)	52/0
7	1062093	rated 52/0	rated 57/0	P 118154	29 1005865	79 Cornershop	100/30 Brimful Of Asha	0/0	49 When I Was Born For The 7th Time *	70/49 (1) (14, 77,)	52/0
8	867248	rated 52/0	rated 40/0	P 95006	16 1021869	79 R.E.M.	100/30 Wake-Up Bomb, The - (live)	0/0	49 New Adventures In Hi-Fi	70/49 (1) (14, 77,)	52/0
9	227025	rated 52/0	rated 48/0	P 25421	42 1006149	79 Crosby, Stills & Nash	100/30	0/0	49 Southern Cross	70/49 (1) CSN [Box]	52/0 (13, 14, 16, 24, 77,)
10	857682	rated 52/0	rated 50/0	P 93944	44 1028256	79 The Verve Pipe	100/30 Cattle	0/0	49 Villains *	70/49 (1) (14, 78,)	52/0
11	1081855	rated 52/0	rated 38/0	N 119843	-1 1024639	79 Sixpence None The Richer	100/30 We Have Forgotten Sixpence None The Richer *	0/0	49 (14, 77,)	70/49 (1)	52/0
12	454986	rated 52/0	rated 46/0	N 50795	-1 1020940	79 Pet Shop Boys	100/30 Heart	0/0	49 Discography - The Complete Singles... (14, 77,)	70/49 (1)	52/0
13	455822	rated 52/0	rated 42/0	P 50895	31 1020983	79 Tom Petty	100/30 Face In The Crowd, A	0/0	49 Full Moon Fever	70/49 (1) (14, 77,)	52/0
14	664522	rated 52/0	rated 47/0	N 73173	-1 1016600	79 Madonna	100/30 Secret	0/0	49 Bedtime Stories	70/49 (1) (7, 14, 24, 76, 77,)	52/0
15	990161	rated 52/0	rated 44/0	N 110565	-1 1027386	79 Train	100/30 Days	0/0 Train	49	70/49 (1) (14, 77,)	52/0
16	544499	rated 52/0	rated 47/0	P 60022	12 1025830	79 The Stone Roses	100/30 Love Spreads	0/0	49 Second Coming	70/49 (1) (14, 77,)	52/0
17	857683	rated 52/0	rated 49/0	N 93944	-1 1028256	79 The Verve Pipe	100/30 Veneer	0/0	49 Villains *	70/49 (1) (14, 78,)	52/0
18	990158	rated 52/0	rated 50/0	N 110565	-1 1027386	79 Train	100/30 Blind	0/0 Train	49	70/49 (1) (14, 77,)	52/0

19	1119487	rated	rated	N	-1	79	100/30	0/0	49	70/49 (1)	52/0
		52/0	55/0	123589	1028125	Various Artists		Block Rockin' Beats - The Chemical Brothers Digital Empire: Electronica's Best (14, 77,)			
20	458446	rated	rated	P	33	79	100/30	0/0	49	70/49 (1)	52/0
		52/0	37/0	51128	1021142	Pink Floyd		Narrow Way Part 1, The - David Gilmour			
		Ummagumma		(14, 77,)							

«entries omitted».

360	830167	rated	rated	N	-1	42	0/0	0/0	42	60/42 (1)	52/0
		52/0	49/0	90869	1016358	Lush	Ladykillers		Lovelife *	(14, 77,)	
#	songID	query comm	origin albumID	status artistID	ord artist	score title	lastP. album	bds	impl.	rating(t)	djs netP.
361	345744	rated	rated	N	-1	42	0/0	0/0	42	60/42 (1)	52/0
		52/0	49/0	38706	1013691	Journey	Faithfully		Time Cubed [Box]		(14, 77,)
362	1012355	rated	rated	N	-1	42	0/0	0/0	42	60/42 (1)	52/0
		52/0	45/0	113423	1023631	Savage Garden		To The Moon & Back		Savage Garden	
		(14, 77,)									
363	673063	rated	rated	N	-1	42	0/0	0/0	42	60/42 (1)	52/0
		52/0	47/0	74082	1025801	Sting	When We Dance - (previously unreleased)				Fields Of Gold: The Best Of Sting 1984-1994 (14, 77,)
364	1383771	rated	rated	N	-1	42	0/0	0/0	42	60/42 (1)	52/0
		52/0	46/0	148392	1021623	The Prodigy	Smack My Bitch Up		Fat Of The Land *	(14, 77,)	
365	499807	rated	rated	N	-1	42	0/0	0/0	42	60/42 (1)	52/0
		52/0	51/0	55333	1023239	Rush	Tom Sawyer		Chronicles	(14, 77,)	
366	1078501	rated	rated	N	-1	42	0/0	0/0	42	60/42 (1)	52/0
		52/0	35/0	119582	1015272	Led Zeppelin	Thank You - (stereo)		BBC Sessions *		
		(14, 77,)									
367	1327003	rated	rated	N	-1	41	0/0	0/0	41	59/41 (1)	52/0
		52/0	43/0	142307	1039472	Tommy Henriksen		Dreaming In Colors		Tommy Henriksen	(14, 77,)
368	1212748	rated	rated	N	-1	40	0/0	0/0	40	57/40 (1)	52/0
		52/0	63/0	132410	1027798	U2	All I Want Is You		The Best Of 1980-1990 [Limited]		(14, 77,)
369	345875	rated	random	N	-1	37	100/30	0/0	7	10/07 (1)	52/0
		52/0	36/0	38717	1013699	Joy Of Cooking	Three Day Loser		American Originals		
		(14, 77,)									
370	1233646	rated	random	N	-1	37	100/30	0/0	7	10/07 (1)	52/0
		52/0	40/0	134584	1037731	Britney Spears	Crazy, (You Drive Me)		Baby One More Time... [ECD]		(14, 77,)
371	573363	rated	random	N	-1	37	100/30	0/0	7	10/07 (1)	52/0
		52/0	40/0	63494	1027743	Twisted Sister	We're Not Gonna Take It		Big Hits And Nasty Cuts-Best Of Twisted Sister		(15, 16,)
372	339153	rated	random	N	-1	37	100/30	0/0	7	10/07 (1)	52/0
		52/0	41/0	37973	1013350	Jethro Tull	Jeffrey Goes To Leicester Square		Stand Up		(14, 77,)
373	1233649	rated	random	N	-1	37	100/30	0/0	7	10/07 (1)	52/0
		52/0	40/0	134584	1037731	Britney Spears	Born To Make You Happy		Baby One More Time... [ECD]		(14, 77,)
374	1411604	rated	random	N	-1	37	100/30	0/0	7	10/07 (1)	52/0
		52/0	43/0	50365	1020680	The Pastels	Baby Honey		Suck On The Pastels...1983-1985		(14, 77,)
375	870674	rated	random	N	-1	37	100/30	0/0	7	10/07 (1)	52/0
		52/0	43/0	95367	1021928	Rage Against The Machine			Year Of Tha Boomerang		
		Evil Empire *		(14, 77,)							

376	1233647	rated	random	N	-1	36	100/30	0/0	6	09/06 (1)	52/0
		52/0	23/0	134584	1037731	Britney Spears		Sometimes		Baby One More Time...	
	[ECD]	(14, 77,)									
377	990162	rated	rated	N	-1	35	0/0	0/0	35	50/35 (1)	52/0
		52/0	39/0	110565	1027386	Train		Rat	Train	(14, 77,)	
378	578086	rated	rated	N	-1	35	0/0	0/0	35	50/35 (1)	52/0
		52/0	49/0	64109	1028073	Van Halen		Top Of The World		For Unlawful	
	Carnal Knowledge		(14, 77,)								
379	948179	rated	rated	N	-1	35	0/0	0/0	35	50/35 (1)	52/0
		52/0	50/0	104678	1015374	Lemonheads		6ix	Car Button Cloth	(14, 77,)	
380	870670	rated	rated	N	-1	35	0/0	0/0	35	50/35 (1)	52/0
		52/0	42/0	95367	1021928	Rage Against The Machine			Down Rodeo	Evil	
	Empire *		(14, 77,)								
381	1327649	rated	rated	N	-1	35	0/0	0/0	35	50/35 (1)	52/0
		52/0	55/0	142358	1003125	Blur		1992	13 [Limited Edition] *	(14, 77,)	
382	1164473	rated	random	N	-1	33	100/30	0/0	3	04/03 (1)	52/0
		52/0	40/0	127996	1017147	John Martyn		Glory Box	The Church With One		
	Bell *	(11,)									
383	1004142	rated	rated	N	-1	31	0/0	0/0	31	44/31 (1)	52/0
		52/0	50/0	112437	1020156	Original Soundtrack			Da Funk - Daft Punk	The	
	Saint	(6,)									
384	1005941	rated	rated	N	-1	28	0/0	0/0	28	40/28 (1)	52/0
		52/0	29/0	112611	1011710	Heart	Stranded	These Dreams - Heart's Greatest Hits *			
		(14, 77,)									
385	531917	rated	rated	N	-1	28	0/0	0/0	28	40/28 (1)	52/0
		52/0	48/0	58747	1025213	Soundgarden		Fell On Black Days		Superunknown	
		(14, 77,)									
386	224547	rated	rated	N	-1	25	0/0	0/0	25	36/25 (1)	52/0
		52/0	45/0	25172	1006025	Crash Test Dummies		Untitled	God Shuffled His Feet		
		(14, 77,)									
387	991308	rated	random	N	-1	21	0/0	0/0	21	30/21 (1)	52/0
		52/0	41/0	110722	1009352	Foo Fighters		New Way Home	The Colour & The Shape		
	*	(14, 78,)									
388	531918	rated	random	N	-1	14	0/0	0/0	14	20/14 (1)	52/0
		52/0	44/0	58747	1025213	Soundgarden		Mailman		Superunknown	(14, 77,
	)										

Implicitly Rated Songs

#	songID	query comm	origin albumID	status artisID	ord artist	score title	lastP. album	bds	impl.	rating(t) djs	netP.
1	559756	random	album	P	6	65	100/20	0/0	45	95/43 (2)	10/1
		52/0	40/2	61860	1026856	They Might Be Giants			Toddler Hiway	They Might Be	
	Giants	(14, 77,)									
2	857672	random	djAlb	P	2	63	100/20	0/0	43	81/36 (2)	90/5
		52/0	36/2	93944	1028256	The Verve Pipe		Drive You Mild	Villains *	(14, 78,	
	)										
3	1212736	djs	album	P	10	61	100/20	0/0	41	80/36 (2)	50/3
		52/0	53/3	132410	1027798	U2		New Year's Day	The Best Of 1980-1990 [Limited]		
		(14, 77,)									
4	1212744	random	album	R	-1	61	100/20	0/0	41	80/36 (2)	40/2
		52/0	61/3	132410	1027798	U2		Sweetest Thing - (The Single mix)		The Best Of	
	1980-1990 [Limited]		(14, 77,)								
5	778854	random	album	R	-1	61	100/20	0/0	41	80/36 (2)	52/3
		52/0	46/2	85650	1024526	Carly Simon		Do The Walls Come Down		Clouds	
	In My Coffee 1965-1995 [Box]		(14, 77,)								
6	778868	random	album	P	8	61	100/20	0/0	41	80/36 (2)	52/3
		52/0	46/2	85650	1024526	Carly Simon		Better Not Tell Her		Clouds In My	
	Coffee 1965-1995 [Box]		(14, 77,)								

7	1089955	random	album	R	-1	61	100/20	0/0	41	80/36 (2)	52/3	
		52/0	45/2	120604	1017716	John Mellencamp	I Need A Lover			The Best That I Could Do...	(14, 77,)	
8	1089962	random	album	R	-1	61	100/20	0/0	41	80/36 (2)	52/3	
		52/0	45/2	120604	1017716	John Mellencamp	Authority Song			The Best That I Could Do...	(14, 77,)	
9	385512	random	album	R	-1	61	100/20	0/0	41	80/36 (2)	50/3	
		52/0	47/2	43305	1016600	Madonna	Papa Don't Preach			The Immaculate Collection	(14, 28, 77,)	
10	778844	random	album	R	-1	61	100/20	0/0	41	80/36 (2)	52/3	
		52/0	42/2	85650	1024526	Carly Simon	Play With Me			Clouds In My Coffee 1965-1995 [Box]	(14, 77,)	
11	778877	random	album	R	-1	61	100/20	0/0	41	80/36 (2)	52/3	
		52/0	42/2	85650	1024526	Carly Simon	Angel From Montgomery - (prev. unreleased)			Clouds In My Coffee 1965-1995 [Box]	(14, 77,)	
12	778855	random	album	R	-1	61	100/20	0/0	41	80/36 (2)	52/3	
		52/0	40/2	85650	1024526	Carly Simon	Danny Boy			Clouds In My Coffee 1965-1995 [Box]	(14, 77,)	
13	1212734	random	album	R	-1	61	100/20	0/0	41	80/36 (2)	50/3	
		52/0	41/2	132410	1027798	U2	Trash, Trampoline And The Party Girl			The Best Of 1980-1990 [Limited]	(14, 77,)	
14	778848	random	album	R	-1	60	100/20	0/0	40	80/36 (2)	52/3	
		52/0	37/2	85650	1024526	Carly Simon	Julie Through The Glass			Clouds In My Coffee 1965-1995 [Box]	(14, 77,)	
15	385563	djs	artist	P	38	60	100/20	0/0	40	80/32 (3)	60/6	
		52/0	49/2	43307	1016600	Madonna	Erotica	Erotica	(14, 77,)			
16	778847	random	album	R	-1	60	100/20	0/0	40	80/36 (2)	52/3	
		52/0	37/2	85650	1024526	Carly Simon	Boys In The Trees			Clouds In My Coffee 1965-1995 [Box]	(14, 77,)	
17	778894	random	album	R	-1	60	100/20	0/0	40	80/36 (2)	52/3	
		52/0	37/2	85650	1024526	Carly Simon	Nobody Does It Better			Clouds In My Coffee 1965-1995 [Box]	(14, 77,)	
18	778890	random	album	R	-1	60	100/20	0/0	40	80/36 (2)	52/3	
		52/0	37/2	85650	1024526	Carly Simon	Why			Clouds In My Coffee 1965-1995 [Box]	(14, 77,)	
19	778856	random	album	R	-1	60	100/20	0/0	40	80/36 (2)	52/3	
		52/0	37/2	85650	1024526	Carly Simon	Dink's Blues			Clouds In My Coffee 1965-1995 [Box]	(14, 77,)	
20	1212752	djs	album	R	-1	60	100/20	0/0	40	80/36 (2)	40/2	
		52/0	48/2	132410	1027798	U2	Love Comes Tumbling			The Best Of 1980-1990 [Limited]	(14, 77,)	

«entries omitted».

2314	1411055	random	random	N	-1	23	100/20	0/0	3	00/00 (4)	0/0	
		52/0	50/3	111845	1026459	Tall Dwarfs	Crocodile			Stumpy *	(14, 77,)	
2315	434293	pop	djArt	N	-1	22	0/0	0/0	22	39/14 (4)	40/6	
		52/0	52/3	48566	1019512	Nine Inch Nails	Ruiner			The Downward Spiral	(14, 77,)	
2316	615943	pop	djArt	N	-1	22	0/0	0/0	22	39/14 (4)	40/6	
		52/0	52/3	68246	1022782	Tom Robinson	Winter Of '79, The			Power In The Darkness	(14, 77,)	
2317	1411059	djs	random	N	-1	22	100/20	0/0	2	00/00 (4)	0/0	
		52/0	42/2	111845	1026459	Tall Dwarfs	Jesus the Beast			Stumpy *	(14, 77,)	
2318	1411054	djs	random	N	-1	22	100/20	0/0	2	00/00 (4)	0/0	
		52/0	40/2	111845	1026459	Tall Dwarfs	The Severed Head of Julio			Stumpy *	(14, 77,)	

2319	1411069	random	random	N	-1	22	100/20	0/0	2	00/00 (4)	0/0	
		52/0	40/2	111845	1026459	Tall Dwarfs		Dessicated		Stumpy *	(14, 77,	
)												
2320	1411070	djs	random	N	-1	22	100/20	0/0	2	00/00 (4)	0/0	
		52/0	40/2	111845	1026459	Tall Dwarfs		Two Minds		Stumpy *	(14, 77,	
)												

#	songID	query comm	origin albumID	status artisID	ord artist	score title	lastP. album	bds	impl.	rating(t) djs	netP.
2321	931183	djs	s avg	N	-1	19	0/0	0/0	19	39/14 (4)	25/4
		52/0	37/2	102305	1012081	Robyn Hitchcock		Yip Song, The		Greatest Hits	(14, 77,
)											
2322	560002	random	random	N	-1	19	0/0	0/0	19	26/09 (4)	52/8
		52/0	47/2	61888	1026872	Thin Lizzy		Killer On The Loose		Life Live	
		(14, 16, 77,)									
2323	1125549	random	artist	N	-1	19	0/0	0/0	19	40/16 (3)	10/1
		52/0	40/2	124176	1023542	Santana	Bella	Best Of Santana (Legacy) *			(14, 77,
)											
2324	328929	random	s avg	N	-1	19	0/0	0/0	19	43/15 (4)	10/2
		52/0	41/2	36886	1012859	INXS	Faith In Each Other			X (14, 77,)	
2325	1073535	djs	s avg	N	-1	18	0/0	0/0	18	46/16 (4)	0/0
		52/0	46/2	119192	1021186	The Pixies		Gouge Away		Death To The Pixies	
		(14, 77,)									
2326	1064098	random	djs	N	-1	18	0/0	0/0	18	26/09 (4)	40/6
		52/0	52/3	118335	1030720	Apollo Four Forty		Ain't Talkin' 'Bout Dub			Electro
Glide In Blue		(14, 43,)									
2327	651483	random	s avg	N	-1	18	0/0	0/0	18	39/14 (4)	10/2
		52/0	47/2	72015	1014381	Carole King		Where You Lead		A Natural Woman: The	
Ode... [Box]		(14, 77,)									
2328	829989	random	s avg	N	-1	17	0/0	0/0	17	39/14 (4)	10/2
		52/0	46/2	90854	1013280	Jefferson Airplane		Crazy Miranda		Bark	(14, 77,
)											
2329	553197	djs	s avg	N	-1	17	0/0	0/0	17	39/14 (4)	10/2
		52/0	44/2	61087	1026455	Talk Talk		Renee		It's My Life (14, 77,)	
2330	651476	djs	s avg	N	-1	17	0/0	0/0	17	39/14 (4)	10/2
		52/0	41/2	72015	1014381	Carole King		I Feel The Earth Move		A Natural	
Woman: The Ode... [Box]		(14, 77,)									
2331	504343	djs	s avg	N	-1	15	0/0	0/0	15	39/14 (4)	0/0
		52/0	34/2	55865	1023614	Joe Satriani		Summer Song		The Extremist	(14, 77,
)											
2332	355176	random	random	N	-1	9	0/0	0/0	9	15/05 (4)	10/2
		52/0	47/2	39927	1014426	The Kinks		Most Exclusive Residence For Sale -			
(mono)		Face To Face		(14, 77,)							
2333	1233652	djs	djs	N	-1	8	0/0	0/0	8	09/04 (2)	40/2
		52/0	41/2	134584	1037731	Britney Spears		I Will Still Love You - (with Don Philip)			
		Baby One More Time... [ECD]		(14, 77,)							
2334	958836	random	random	N	-1	7	0/0	0/0	7	09/03 (4)	10/2
		52/0	37/2	105851	1029091	The Who		I Don't Even Know Myself			Live At
The Isle Of Wight Festival 1970 *		(14, 77,)									

Unrated Songs

#	songID	query comm	origin albumID	status artisID	ord artist	score title	lastP. album	bds	impl.	rating(t) djs	netP.
1	1011924	random	djAlb	P	7	54	100/25	0/0	29	52/00 (0)	73/24
		52/0	46/5	113337	1028125	Various Artists		Crockett's Theme - Jan Hammer			Pure
Moods		(10,)									
2	1011928	random	djAlb	P	11	53	100/25	0/0	28	52/00 (0)	73/24
		52/0	41/4	113337	1028125	Various Artists		Theme From "Twin Peaks - Fire Walk With			
Me" - Angelo Badalamenti					Pure Moods		(10,)				

3 423652 pop random P 17 47 100/25 0/0 22 52/00 (0) 52/17
52/0 52/5 47344 1018869 Peter Murphy Light Pours Out Of Me, The Should
The World Fail To Fall Apart (14, 77,)

4 423656 pop random P 34 47 100/25 0/0 22 52/00 (0) 52/17
52/0 52/5 47344 1018869 Peter Murphy God Sends Should The World Fail To
Fall Apart (14, 77,)

5 1193855 pop random P 37 47 100/25 0/0 22 52/00 (0) 52/17
52/0 52/5 130669 1009757 Fun Lovin' Criminals Big Night Out 100% Columbian
* (14, 77,)

6 423649 random random R -1 47 100/25 0/0 22 52/00 (0) 52/17
52/0 52/5 47344 1018869 Peter Murphy Final Solution Should The World Fail To
Fall Apart (14, 77,)

7 1259512 random random P 45 47 100/25 0/0 22 52/00 (0) 52/17
52/0 52/5 137097 1028125 Various Artists Portnawack - Typhoon Made On Earth
(14, 77,)

8 1259519 random random P 32 47 100/25 0/0 22 52/00 (0) 52/17
52/0 52/5 137097 1028125 Various Artists Untitled - Total Eclipse Made On Earth
(14, 77,)

9 423657 pop random N -1 47 100/25 0/0 22 52/00 (0) 52/17
52/0 52/5 47344 1018869 Peter Murphy Blue Heart Should The World Fail To
Fall Apart (14, 77,)

10 958997 random random N -1 47 100/25 0/0 22 52/00 (0) 52/17
52/0 52/5 105874 1028125 Various Artists Freelon - Spacetime Continuum Werks
Like a Twelve Inch (14, 77,)

11 1193846 pop random N -1 47 100/25 0/0 22 52/00 (0) 52/17
52/0 52/5 130669 1009757 Fun Lovin' Criminals View Belongs To Everyone, The
100% Columbian * (14, 77,)

12 1193848 pop random N -1 47 100/25 0/0 22 52/00 (0) 52/17
52/0 52/5 130669 1009757 Fun Lovin' Criminals Back On The Block 100%
Columbian * (14, 77,)

13 1193844 pop random N -1 47 100/25 0/0 22 52/00 (0) 52/17
52/0 52/5 130669 1009757 Fun Lovin' Criminals Up On The Hill 100% Columbian
* (14, 77,)

14 1193845 random random N -1 47 100/25 0/0 22 52/00 (0) 52/17
52/0 52/5 130669 1009757 Fun Lovin' Criminals Love Unlimited 100% Columbian
* (14, 77,)

15 923902 random random N -1 47 100/25 0/0 22 52/00 (0) 52/17
52/0 52/5 101415 1028125 Various Artists Grass Roots - Tricky/Roberto Malary Jr.
Tricky Presents Grassroots [EP] (14, 77,)

16 1193854 pop random N -1 47 100/25 0/0 22 52/00 (0) 52/17
52/0 52/5 130669 1009757 Fun Lovin' Criminals All My Time Is Gone 100%
Columbian * (14, 77,)

17 1193849 pop random N -1 47 100/25 0/0 22 52/00 (0) 52/17
52/0 52/5 130669 1009757 Fun Lovin' Criminals 10th Street 100% Columbian
* (14, 77,)

18 1193852 pop random N -1 47 100/25 0/0 22 52/00 (0) 52/17
52/0 52/5 130669 1009757 Fun Lovin' Criminals We Are All Very Worried About
You 100% Columbian * (14, 77,)

19 806170 random random N -1 47 100/25 0/0 22 52/00 (0) 52/17
52/0 52/5 88136 1028125 Various Artists Man's World, (It's Not) A - Strata 3 The Trip
Hop Test Part 2 (14, 77,)

20 806163 random random N -1 47 100/25 0/0 22 52/00 (0) 52/17
52/0 52/5 88136 1028125 Various Artists Anafey - Hip Optimist The Trip Hop
Test Part 2 (14, 77,)

«entries omitted».

1304 228812 pop random N -1 22 0/0 0/0 22 52/00 (0) 52/17
52/0 52/5 25620 1030126 The Crystals I Wonder The Best Of The Crystals
(23,)

1305	228814	pop	random	N	-1	22	0/0	0/0	22	52/00 (0)	52/17
		52/0	52/5	25620	1030126	The Crystals		Girls Can Tell		The Best Of The Crystals	
		(23,)									
1306	228798	pop	random	N	-1	22	0/0	0/0	22	52/00 (0)	52/17
		52/0	52/5	25620	1030126	The Crystals		Oh, Yeah, Maybe, Baby		The Best Of The	
	Crystals (23,)										
1307	228810	random	random	N	-1	22	0/0	0/0	22	52/00 (0)	52/17
		52/0	52/5	25620	1030126	The Crystals		Heartbreaker		The Best Of The Crystals	
		(23,)									
1308	740607	pop	random	N	-1	22	0/0	0/0	22	52/00 (0)	52/17
		52/0	52/5	81532	1008091	EBN		Get Down Ver. 2.2		Telecommunication	
	Breakdown [ECD]			(14, 77,)							
1309	876063	pop	random	N	-1	22	0/0	0/0	22	52/00 (0)	52/17
		52/0	52/5	95946	1012421	Howie B.		Shag		Music For Babies	(14, 77,)
1310	914734	pop	random	N	-1	22	0/0	0/0	22	52/00 (0)	52/17
		52/0	52/5	100059	1020939	Pet		Fatherland		Pet	(14, 77,)
1311	882981	pop	random	N	-1	22	0/0	0/0	22	52/00 (0)	52/17
		52/0	52/5	96691	1028125	Various Artists		Million Town - Strange Cargo (The Kruder			
	& Dorfmeister Session)			A Journey Into Ambient Groove 3				(14, 77,)			
1312	1320082	pop	random	N	-1	22	0/0	0/0	22	52/00 (0)	52/17
		52/0	52/5	141627	1039729	Papa Vegas		Something Wrong		Hello Vertigo	
	[4/27]	(14, 77,)									
1313	1242704	pop	random	N	-1	22	0/0	0/0	22	52/00 (0)	52/17
		52/0	52/5	135883	1038686	The Hope Blister		Hanky Panky Nohow		Smile's OK...	
		(14, 77,)									
1314	942415	random	random	N	-1	22	0/0	0/0	22	52/00 (0)	52/17
		52/0	52/5	103598	1024664	Skeleton Key		World's Most Famous Undertaker, The			
	Skeleton Key [EP]			(14, 77,)							
1315	1119500	pop	random	N	-1	22	0/0	0/0	22	52/00 (0)	52/17
		52/0	52/5	123589	1028125	Various Artists		Take California - Propellerheads			Digital
	Empire: Electronica's Best			(14, 77,)							
1316	528565	pop	random	N	-1	22	0/0	0/0	22	52/00 (0)	52/17
		52/0	52/5	58464	1025129	Sons Of Champlin			Get High		Capitol Gold:
	The Best Of The Sons Of Champlin			(14, 77,)							
1317	528568	pop	random	N	-1	22	0/0	0/0	22	52/00 (0)	52/17
		52/0	52/5	58464	1025129	Sons Of Champlin			It's Time	Capitol Gold: The Best Of	
	The Sons Of Champlin			(14, 77,)							
1318	942223	random	random	N	-1	22	0/0	0/0	22	52/00 (0)	52/17
		52/0	52/5	103571	1024799	Sloan		G Turns To D		One Chord To Another	(14, 77,
)											
1319	942219	random	random	N	-1	22	0/0	0/0	22	52/00 (0)	52/17
		52/0	52/5	103571	1024799	Sloan		Good In Everyone, The		One Chord To Another	
		(14, 77,)									
1320	1017638	random	random	N	-1	22	0/0	0/0	22	52/00 (0)	52/17
		52/0	48/5	114082	1004159	David Byrne		Wicked Little Doll		Fcelings *	
		(14, 77,)									

#	songID	query comm	origin albumID	status artistID	ord artist	score title	lastP. album	bds	impl.	rating(t)	djs	netP.
1321	809747	random	random	N	-1	22	0/0	0/0	22	52/00 (0)		52/17
		52/0	46/5	88473	1015875	Loop Guru		Jungle ADuniya		(14, 77,)		
1322	455363	random	random	N	-1	21	0/0	0/0	21	52/00 (0)		52/17
		52/0	40/4	50841	1030292	Peter & Gordon		I Feel Like Going Out		The Best Of		
	Peter & Gordon (Rhino)			(23,)								
1323	814350	random	djArt	N	-1	18	0/0	0/0	18	52/00 (0)		40/13
		52/0	45/5	88938	1021734	Pulp		Death II Separations		(14, 77,)		
1324	232378	djs	random	N	-1	12	0/0	0/0	12	52/00 (0)		20/7
		52/0	49/5	26074	1006547	The Damned		Smash It Up (Parts 1 & 2)		The Best Of The		
	Damned (Another...)			(14, 78,)								

</PRE>
<XMP><ASX VERSION="3.0" PREVIEWMODE="NO">

```
<REPEAT>
  <ENTRY>
    <REF HREF="http://devweb7.launch.com/servlet/gateway?u=6474126&n=0.asp"/>
  </ENTRY>
  <ENTRY>
    <REF HREF="http://devweb7.launch.com/servlet/gateway?u=6474126&n=1.asp"/>
  </ENTRY>
  <ENTRY>
    <REF HREF="http://devweb7.launch.com/servlet/gateway?u=6474126&n=2.asp"/>
  </ENTRY>
  <ENTRY>
    <REF HREF="http://devweb7.launch.com/servlet/gateway?u=6474126&n=3.asp"/>
  </ENTRY>
  <ENTRY>
    <REF HREF="http://devweb7.launch.com/servlet/gateway?u=6474126&n=4.asp"/>
  </ENTRY>
  <ENTRY>
    <REF HREF="http://devweb7.launch.com/servlet/gateway?u=6474126&n=5.asp"/>
  </ENTRY>
  <ENTRY>
    <REF HREF="http://devweb7.launch.com/servlet/gateway?u=6474126&n=6.asp"/>
  </ENTRY>
  <ENTRY>
    <REF HREF="http://devweb7.launch.com/servlet/gateway?u=6474126&n=7.asp"/>
  </ENTRY>
  <ENTRY>
    <REF HREF="http://devweb7.launch.com/servlet/gateway?u=6474126&n=8.asp"/>
  </ENTRY>
  <ENTRY>
    <REF HREF="http://devweb7.launch.com/servlet/gateway?u=6474126&n=9.asp"/>
  </ENTRY>
</REPEAT>
</ASX>
</XMP>
```

SOURCE CODE

Internet Radio and Broadcast Method
Copyright © 1999, 2000 LAUNCH Media, Inc.
www.LAUNCH.com

ALBUMARTISTDATA 4
ALBUMINFO 5
ARTISTINFO 7
AVERAGERATING 8
BANDWIDTH 9
BDSRANK 11
CACHEDRATING 12
CLIP 13
CLIPCOLLECTION 17
CLIPSCHEDULE 18
CONSTANTS 21
DBCONNECTION 23
DBEXCEPTION 26
DBPREPAREDSTATEMENT 27
DBRESULTSET 28
DJ 31
DJLIST 32
FREQUENCYCOUNTER 34
GENERATORPARAMETERS 37
GENREINDEX 39
GENRELIST 41
GETADS 43
GETBDSSTATIONS 45
GETGENRES 46
GETITEMRATINGSFROMDB 47
GETLASTPLAYED 48
GETNEWS 49
GETPLAYLIST 51
GETPLAYLISTSERVERS 52
GETPLAYLISTSERVERSINTERFACE 53

GETPOPULAR 54
GETRATINGS 55
GETRATINGSCACHEUSERS 59
GETRATINGSCACHEUSERSINTERFACE 61
GETRECENTLYPLAYED 62
GETSONGINFOSERVLET 64
GETSONGRATINGSFROMDB 70
INTHASH 71
ITEM 72
ITEMSPROFILE 74
MEDIA 76
MEDIAFORMAT 77
MEDIAGATEWAYSERVLET 78
MEDIALIST 83
PICKCOUNT 85
PICKLIST 87
PICKSTATUS 88
PLAYDATAHASH 89
PLAYDATES 90
PLAYLIST 98
PLAYLIST2 105
PLAYLISTCREATORTEST 106
PLAYLISTENTRY 107
PLAYLISTGENERATOR 108
PLAYLISTGENERATORSERVLET 120
PLAYLISTMAKER 125
PLAYLISTPARAMETERS 126
PLAYLISTSTATUS 127
POPULARSONGS 130
POPULATION 131
RATING 139
RATINGSCACHE 140
RATINGSPROFILE 146
RATINGWIDGETSERVLET 147

RECLIST 153
SAVECLIPS 156
SAVEPLAYLIST 158
SIMPLECLIP 160
SIMPLECLIPLIST 161
SIMPLEPLAYLIST 162
SONG 165
SONGDATA 167
SONGGROUP 174
SONGINFO 175
SONGINFOCACHE 178
SONGINFOCACHEUPDATER 185
SONGLIST 186
SONGRATING 189
STATION 190
STATIONLIST 191
UTIL 192
WEIGHTMATRIX 194

AlbumArtistData

```
package com.launch.PlaylistGenerator;
public class AlbumArtistData
{
        Item album = null;
        Item artist = null;

        boolean alreadyTriedAlbum = false;
        boolean alreadyTriedArtist = false;

        public void reset()
        {
                album = null;
                artist = null;
                alreadyTriedAlbum = false;
                alreadyTriedArtist = false;
        }

        public Item getAlbum(ItemsProfile items, SongData data)
        {
                if (alreadyTriedAlbum)
                        return album;

                alreadyTriedAlbum = true;

                album = items.get(data.getAlbumID());

                return album;
        }
        public Item getArtist(ItemsProfile items, SongData data)
        {
                if (alreadyTriedArtist)
                        return artist;

                alreadyTriedArtist = true;

                artist = items.get(data.getArtistID());

                return artist;
        }
}
```

AlbumArtistData.java Page 1 of 1 11/05/99 1:32 PM

AlbumInfo

```
package com.launch.PlaylistGenerator;
import java.util.Vector;
public class AlbumInfo
{
	int ID;
	String title;
	ArtistInfo artist;

	Vector genres;
	public AlbumInfo(int ID)
	{
		this.ID = ID;
	}

	public String toString()
	{
		return "[albumID=" + ID + ", title=" + title
			+ ", genres=" + genresString() + ", artist=" + artist.toString() + "]";
	}

	public String genresString()
	{
		if (genres == null)
			return "(NONE)";

		String result = "";

		for (int i = 0; i < genres.size(); i++)
		{
			result = result.concat(genres.elementAt(i) + ", ");
		}
		return "(" + result + ")";
	}

	public int getArtistID() throws Exception
	{
		if (artist == null)
			throw new Exception("artist is not set for album " + ID + " (" + title + ") ");

		return artist.ID;
	}

	public boolean inGenres(short genreID)
	{
		if (genres == null)
			return false;

		return genres.contains(new Short(genreID));

	}

	public boolean inGenres(GenreList userGenres)
	{

		if (userGenres.allGenres == true)
			return true;

		if (genres == null)
			return false;
```

ArtistInfo

```
package com.launch.PlaylistGenerator;
import java.util.Hashtable;
public class ArtistInfo
{
        int ID;
        String title;
        Hashtable songs;

        public ArtistInfo(int ID)
        {
                this.ID = ID;
                songs = new Hashtable();

        }

        public String toString()
        {
                return "[artistID=" + ID + ", title=" + title + "]";
        }
        public final static boolean isVariousArtists(int itemID)
        {
                return (itemID == Constants.ARTIST_VARIOUS_ARTISTS
                        || itemID == Constants.ARTIST_ORIGINAL_SOUNDTRACK
                        || itemID == Constants.ARTIST_SOUNDTRACK);
        }

}
```

ArtistInfo.java Page 1 of 1 11/05/99 1:37 PM

```
                // do it the other way, check each of the genres the song is
                // in and if it's in the user's genres

                for (int i = 0; i < genres.size(); i++)
                {
                        Short genreID = (Short) genres.elementAt(i);

                        if (userGenres.exists(genreID))
                                return true;
                }

                return false;
        }

        public void addGenre(short genreID)
        {

                if (genres == null)
                        genres = new Vector(1,1);

                // be careful not to add duplicates
                Short genre = new Short(genreID);

                if (!genres.contains(genre))
                        genres.addElement(new Short(genreID));

        }

}
```

AlbumInfo.java Page 2 of 2 11/05/99 1:27 PM

AverageRating

```
package com.launch.PlaylistGenerator;
public class AverageRating extends Rating
{
    private short count = 0;
    private int sum;
    private boolean calculated = false;
    public AverageRating()
    {
        super();
    }
    public AverageRating(short defaultRating)
    {
        super(defaultRating);
    }
    public void add(int value)
    {
        sum += value;
        count++;
        calculated = false;
    }
    public short get()
    {
        calculate();
        return super.get();
    }
    public short count()
    {
        return count;
    }
    private void calculate()
    {
        if (!calculated)
        {
            if (count > 0)
            {
                set(Util.average(count, sum));
                set = true;
            }
            calculated = true;
        }
    }
    public String toString()
    {
        String ratingStr = "(Not calculated)";
        if (set) ratingStr = "" + rating;
        return sum + "/" + count + "=" + ratingStr;
    }
}
```

AverageRating.java Page 2 of 2 11/05/99 1:27 PM

Bandwidth

```
package com.launch.PlaylistGenerator;
public class Bandwidth
{
        public final static short SPEED_28  = 28;
        public final static short SPEED_56  = 56;
        public final static short SPEED_100 = 100;
        public final static short SPEED_128 = 128;
        public final static short SPEED_300 = 300;
        public final static short SPEED_500 = 500;

        private boolean beenset = false;
        private short value = SPEED_28;

        public Bandwidth()
        {

        }

        public Bandwidth(short speed)
        {
                value = speed;
                beenset = true;
        }

        public Bandwidth(String speed)
        {
                if (speed == null)
                {
                        beenset = false;
                }
                else
                {
                        if (speed.equals("28"))
                                set(SPEED_28);
                        else if (speed.equals("56"))
                                set(SPEED_56);
                        else if (speed.equals("100"))
                                set(SPEED_100);
                        else if (speed.equals("128"))
                                set(SPEED_128);
                        else if (speed.equals("300"))
                                set(SPEED_300);
                        else if (speed.equals("500"))
                                set(SPEED_500);
                        else
                        {
                                beenset = false;
                        }
                }
        }

        public String toString()
        {

                if (value == SPEED_28)
                        return "28.8k";
                else if (value == SPEED_56)
                        return "56k";
                else if (value == SPEED_100)
```

```
                return "100k";
            else if (value == SPEED_128)
                return "128k";
            else if (value == SPEED_300)
                return "300k";
            else if (value == SPEED_500)
                return "56k";
            return "UNKNOWN (" + value + ")";
        }

        public short get()
        {
            return value;
        }

        public void set(short speed)
        {
            if (speed == SPEED_28
                || speed == SPEED_56
                || speed == SPEED_100
                || speed == SPEED_128
                || speed == SPEED_300
                || speed == SPEED_500)
            {
                value = speed;
                beenset = true;
            }
            else
                beenset = false;
        }

        public boolean load(DBConnection conn, int userID)
        {
            try
            {
                DBResultSet rs = conn.executeSQL("exec sp_a150UserPreference_GetValue_xsxx " +
userID);
                if (!rs.getBOF() && !rs.getEOF())
                {
                    set(rs.getShort("iDefaultBandwidth"));
                }
            }
            catch (DBException oops)
            {
                Util.debug("DB Exception in Bandwidth::load: " + oops.getMessage());
            }

            return isSet();

        }

        public boolean isSet()
        {
            return beenset;
        }
}
```

BDSRank

```
package com.launch.PlaylistGenerator;
public class BDSRank
{
        short stationID;
        byte rank;

        public BDSRank(short stationID, byte rank)
        {
                this.stationID = stationID;
                this.rank = rank;
        }

        public String toString()
        {
                return stationID + ":" + rank;
        }

}
```

BDSRank.java Page 1 of 1 11/05/99 1:26 PM

CachedRating

```
package com.launch.PlaylistGenerator;
import java.io.*;
import java.util.Date;
/**
 * This class is used to model a single rating in the cache.
 **/
public final class CachedRating implements Serializable
{
        public int userID;
        public int itemID;
        public byte rating;
        public byte type;
        private Date created = new Date();

    //---------------------------------------------------------
        public CachedRating(int userID, int itemID, byte rating, byte type)
        {
                this.userID = userID;
                this.itemID = itemID;
                this.rating = rating;
                this.type   = type;
        }
        public final String toString()
        {
                return("user:" + userID + ", itemID:" + itemID + ", rating:" + rating + ", type:" +
typeString(type) + ", date:" + created.toString() + Util.newLine);
        }

        public final static String typeString(byte type)
        {
                if (type == Constants.ITEM_TYPE_SONG)
                        return "song";
                else if (type == Constants.ITEM_TYPE_ALBUM)
                        return "album";
                else if (type == Constants.ITEM_TYPE_ARTIST)
                        return "artist";
                return "unknown";
        }

        public String hashKey()
        {
                return itemID + ":" + type;
        }
}
```

CachedRating.java Page 1 of 1 11/05/99 1:35 PM

Clip

```
package com.launch.PlaylistGenerator;
import java.util.Date;
public class Clip
{
        public final static byte TYPE_NONE = 0;
        public final static byte TYPE_NEWS = 1;
        public final static byte TYPE_AD  = 2;
        public final static byte TYPE_INTERSTITIAL = 3;
        public final static byte TYPE_TIP = 4;
        public final static byte TYPE_SONG = 5;
        public final static byte TYPE_BROADCAST = 6;

        public int ID;
        public byte type;
        public int mediaID;
        public Date lastPlayed;
        public String name, directory, server, filepath;
        public MediaList media;
        byte origin;

        private boolean set = false;
        public Clip(byte type)
        {
                this.type = type;
                media = new MediaList();
        }
        public Clip(int ID, byte type)
        {
                this(type);
                this.ID = ID;
        }

        public Clip(int ID, byte type, int mediaID, String name, Date lastPlayed)
        {
                this(ID, type);
                this.ID     = ID;
                this.mediaID   = mediaID;
                this.name      = name;
                this.lastPlayed = lastPlayed;
        }

        public byte type() { return type; }

        public boolean isSet() { return set; }

        private void setDirectory(String newDir)
        {
                if (!newDir.equals(" "))
                {
                        directory = newDir;
                }
        }
        public void logPlay(DBConnection conn, int userID)
        {
                String sql = "";

                if (type == TYPE_SONG)
                        sql = "exec sp_lcLogPlaySong_isud "     + userID + ", " + mediaID + ", " + ID + ", " +
origin;
```

```
                else if (type == TYPE_AD)
                        sql = "exec sp_lcLogPlayAd_isud "       + userID + ", " + mediaID + ", " + ID;
                else if (type == TYPE_NEWS)
                        sql = "exec sp_lcLogPlayNews_isud "     + userID + ", " + mediaID + ", " + ID;
                else if (type == TYPE_TIP)
                        sql = "exec sp_lcLogPlayTip_isud "      + userID + ", " + mediaID + ", " + ID;
//              else if (type == TYPE_BROADCAST)
//                      sql = "exec sp_lcLogPlayBroadcast_isux " + userID + ", " + mediaType;

                try
                {
                        conn.executeUpdate(sql, true);
                }
                catch (DBException e)
                {
                        System.err.println("DBException in Clip:logPlay:" + e.toString());
                }

        }

        public boolean getPath(DBConnection conn, ClipSchedule schedule)
        {
                if (type == TYPE_NONE)
                        return false;

                SimpleClipList list = null;

                if (type == TYPE_SONG)
                        list = schedule.playlist.songs;
                else if (type == TYPE_AD)
                        list = schedule.playlist.ads;
                else if (type == TYPE_TIP)
                        list = schedule.playlist.tips;
                else if (type == TYPE_NEWS)
                        list = schedule.playlist.news;

                if (list == null)
                        return false;

                SimpleClip yip = list.pop();

                if (yip == null)
                        return false;

                mediaID = yip.mediaID;
                ID  = yip.ID;
                origin  = yip.origin;


                try
                {
                        DBResultSet rs = conn.executeSQL("exec sp_lcGetMediaPath_xsxx " + mediaID);

                        if (!rs.getBOF() && !rs.getEOF())
                        {
                                setDirectory(rs.getString("directory"));
                                server   = rs.getString("server");
                                filepath = rs.getString("filepath");

                                set = true;
                        }
                }
```

```
                catch (DBException e)
                {
                        System.err.println("DBException in Clip::getPath: " + e.toString());
                }

                return set;
        }
        /*
        public boolean pop(DBConnection conn, int userID, int context)
        {
                set = false;
                try
                {
                        DBResultSet rs;
                        String the_command;

                        int contextNum = 0;
                        if (context > 1) contextNum = 1;

                        if (type==TYPE_BROADCAST)
                        {
                                the_command="exec " + BROADCAST_SP + " " + userID + ", " + type + ", " +
context;
                        }
                        else
                        {
                                String stored_proc = null;
                                if     (type == TYPE_AD  ) stored_proc = ADS_SP;
                                else if (type == TYPE_TIP ) stored_proc = TIPS_SP;
                                else if (type == TYPE_NEWS) stored_proc = NEWS_SP;
                                else                        stored_proc = SONG_SP;
                                the_command= "exec " + stored_proc + " " + userID + ", " + contextNum;
                        }
                        rs = conn.executeSQL(the_command);
                        if (!rs.getBOF() && !rs.getEOF())
                        {
                                setDirectory(rs.getString("directory"));
                                server   = rs.getString("server");
                                filepath = rs.getString("filepath");

                                set = true;
                        }
                }
                catch (DBException e)
                {
                        System.err.println("DBException in Clip::pop: " + e.toString());
                }

                return isSet();
        }
        */

        public String path()
        {

                return server
                        + directory
                        + "/"
                        + filepath;
        }

        public String toString()
```

```
185         {
                return "Clip type (" + typeName() + "), id = " + mediaID
                        + ", lastPlayed = " + lastPlayed
                        + ", media = " + media.toString()
                        + ", path = " + path();
190
        }

        public PlaylistEntry toPlaylistEntry(short mediaType)
        {
195
                PlaylistEntry entry = new PlaylistEntry();
                entry.mediaID = media.getID(mediaType);
                entry.title   = name;

200             entry.filepath = media.getFilepath(mediaType);

                return entry;
        }

205     public SimpleClip toSimpleClip(short mediaType)
        {
                return new SimpleClip(ID, media.getID(mediaType));
        }

210     public String typeName()
        {
                switch(type)
                {
                case TYPE_AD:
215                     return "Ad";
                case TYPE_BROADCAST:
                        return "Broadcast";
                case TYPE_INTERSTITIAL:
                        return "Interstitial";
220             case TYPE_NEWS:
                        return "News";
                case TYPE_TIP:
                        return "Tip";
                case TYPE_SONG:
225                     return "Song";
                }

                return "?";
        }
230
        public String URL()
        {

                return server
235                     + directory
                        + "/"
                        + filepath;
        }

240 }
```

ClipCollection

```
package com.launch.PlaylistGenerator;
import java.util.Hashtable;
public class ClipCollection extends Hashtable
{
        public Clip put(int clipID, Clip aClip)
        {
                return (Clip) put(new Integer(clipID), aClip);
        }

        public Clip get (int clipID)
        {
                return (Clip) get(new Integer(clipID));
        }
}
```

ClipCollection.java Page 1 of 1 11/05/99 1:26 PM

ClipSchedule

```
package com.launch.PlaylistGenerator;
import java.util.Date;
import javax.servlet.ServletOutputStream;
public class ClipSchedule
{
        private Date dbDate;

        private int userID, lastBroadcast, currentBroadcast;

        private boolean set = false;
        public SimplePlaylist playlist;

        public ClipSchedule (int userID)
        {
                this.userID = userID;
        }

        public void init(DBConnection conn)
        {

                set = false;
                try
                {
                        DBResultSet rs = conn.executeSQL("exec sp_lcGetClipSchedule_xsxx " + userID);
                        if (!rs.getBOF() && !rs.getEOF())
                        {
                                dbDate        = rs.getTimestamp("dbDate");
                                lastBroadcast  = rs.getInt("lastBroadcastID");
                                currentBroadcast = rs.getInt("broadcastID");
                                playlist      = SimplePlaylist.fromBytes(rs.getBytes("playlist"));

                        }
                        else
                        {
                                dbDate = new Date();
                        }

                        // the first time a playlist is created for a user, the dates will be null

                        if (playlist != null)
                        {
                                if (playlist.lastAd  == null) playlist.lastAd  = dbDate;
                                if (playlist.lastNews == null) playlist.lastNews = dbDate;
                                if (playlist.lastTip  == null) playlist.lastTip  = dbDate;
                                set = true;
                        }
                }
                catch (DBException e)
                {
                        System.err.println("DBException in ClipSchedule::init:" + e.toString());
                }
        }

        private long dateDiff(Date diffMe)
        {

                if (diffMe == null)
                        diffMe = new Date(0);
```

```
            return (long) ((dbDate.getTime() - diffMe.getTime()) / (1000.0 * 60));
        }

        public byte nextClipType(boolean debug, ServletOutputStream out)
        {

            long adDiff, newsDiff, tipDiff;

            while (true)
            {
                adDiff  = dateDiff(playlist.lastAd);
                newsDiff = dateDiff(playlist.lastNews);
                tipDiff  = dateDiff(playlist.lastTip);

                if (debug)
                {
                    Util.out(out, "dbDate is " + dbDate.toString());

                    Util.out(out, "lastAdDate is " + playlist.lastAd);
                    Util.out(out, "next ad in " + (Constants.AD_THRESHOLD - adDiff) + "
minutes");

                    Util.out(out, "lastNewsDate is " +  playlist.lastNews);
                    Util.out(out, "next news clip in " + (Constants.NEWS_THRESHOLD -
newsDiff) + " minutes");

                    Util.out(out, "lastTipDate is " +  playlist.lastTip);
                    Util.out(out, "next tip in  " + (Constants.TIP_THRESHOLD - tipDiff) + "
minutes");
                }

                if (playlist == null)
                {
                    System.err.println(new Date().toString() + " nextClipType: userID " + userID +
" has no/invalid playlist");
                    return Clip.TYPE_NONE;
                }

                if (currentBroadcast > lastBroadcast)
                {
                    if (debug) Util.out(out, "getting broadcast");
                    lastBroadcast = currentBroadcast;
                    return Clip.TYPE_BROADCAST;
                }
                else if (adDiff >= Constants.AD_THRESHOLD)
                {
                    if (debug) Util.out(out, "playing AD");
                    playlist.lastAd = dbDate;

                    if (playlist.ads.isEmpty())
                        System.err.println(new Date().toString() + " userID " + userID + " is
out of ads");
                    else
                        return Clip.TYPE_AD;
                }
                else if (newsDiff >= Constants.NEWS_THRESHOLD)
                {
                    if (debug) Util.out(out, "playing NEWS");
                    playlist.lastNews = dbDate;

                    if (playlist.news.isEmpty())
                        System.err.println(new Date().toString() + " userID " + userID + " is
```

```
out of news");
                                        else
                                                return Clip.TYPE_NEWS;
                                }
                                else if (tipDiff >= Constants.TIP_THRESHOLD)
                                {

                                        if (debug) Util.out(out, "playing TIP");
                                        playlist.lastTip = dbDate;

                                        if (playlist.tips.isEmpty())
                                                System.err.println(new Date().toString() + " userID " + userID + " is
out of tips");
                                        else
                                                return Clip.TYPE_TIP;
                                }
                                else
                                {
                                        if (debug) Util.out(out, "playing SONG");

                                        if (playlist.songs.isEmpty())
                                        {
                                                System.err.println(new Date().toString() + " userID " + userID + " is
out of songs");
                                                return Clip.TYPE_NONE;
                                        }
                                        else
                                                return Clip.TYPE_SONG;
                                }

                        }
                        //return Clip.TYPE_NONE;
                }
        }
ClipSchedule.java       Page 3 of 3     11/05/99 1:35 PM
```

Constants

```
package com.launch.PlaylistGenerator;
public interface Constants
{
        // live
        /*
        public final static String DB_SOURCE            = "LAUNCHcast";
        public final static String DB_USERNAME          = "dbClient";
        public final static String DB_PASSWORD          = "83kareem23";
        public final static String DB_DBNAME            = "dbLaunchProd";
        public final static String DB_SERVER            = "209.67.158.19"; // DB3
        public final static short  DB_PORT              = 1433;
        public final static String STREAM_URL = "http://lcplaylist.launch.com/servlet/gateway";
        public final static String STREAM_SERVER = "http://lcstream.launch.com";
        */
        // development
        public final static String DB_SOURCE            = "LAUNCHcast";
        public final static String DB_USERNAME          = "dbClient";
        public final static String DB_PASSWORD          = "29Idiocy99";
        public final static String DB_DBNAME            = "dbLaunchProd";
        public final static String DB_SERVER            = "zeus";
        public final static short  DB_PORT              = 1433;
        public final static String STREAM_URL = "http://devweb7.launch.com/servlet/gateway";
        public final static String STREAM_SERVER = "http://devweb7.launch.com/F";
        public final static int RIAA_MAX_SONGS_FROM_ALBUM    = 2;
        public final static int RIAA_MAX_SONGS_BY_ARTIST     = 3;
        public final static int BDS_SCORE_MAX_POINTS         = 41;
        public final static int BDS_SCORE_POINTBAR           = 20;
        public final static int DEFAULT_LASTPLAYED_SCORE             = 100;
        public final static int DEFAULT_MEDIATYPE                    = 211; // 16 Mono
        public final static int DEFAULT_UNRATED_RATIO        = 50;
        public final static int DEFAULT_PICK_FACTOR          = 7;
        public final static int DEFAULT_BDS_SCORE                    = 0;
        public final static int MAX_PERCENT_RATED_SONGS_TO_PICK = 20;
        public final static int NEW_USER_UNRATED_RATIO       = 90;
        public final static int MIN_RATINGS_TO_HONOR_RATIO   = 100;
        public final static int MIN_SIZE_FOR_NO_UNRATED      = 200;
        public final static int MAX_ORDINAL                          = 1000;
        // for calculating implicit based on other song ratings
        public final static int MAX_SONGS_BY_ARTIST                  = 4;
        // random picking
        public final static int RANDOM_SONGS_COUNT           = 5000;
        // this is a percent of the total number of songs in the database
        public final static int MIN_SONGS_IN_GENRES_TO_GET_RANDOM = 5;
        public final static int MIN_RATING_FOR_RATED_SOURCE          = 35;
        // songs with average rating above this are considered popular
        // also change this at the top of LAUNCHCast/player/getsonginfo
        public final static int POPULAR_THRESHOLD                    = 58;
        public final static int DEFAULT_RATING                       = 52; // global average for
all songs
        public final static int DEFAULT_DJS_SCORE                    = DEFAULT_RATING;
        public final static int DEFAULT_NETP_SCORE                   = DEFAULT_RATING;
        public final static byte DEFAULT_COMMRATING                  = DEFAULT_RATING;
        public final static int MAX_RATINGS_TO_GET                   = 500;
        public final static int MAX_DJ_RATINGS_TO_GET                = 500;
        public final static int ARTIST_VARIOUS_ARTISTS       = 1028125;
        public final static int ARTIST_ORIGINAL_SOUNDTRACK           = 1020156;
        public final static int ARTIST_SOUNDTRACK                    = 1036715;
        public final static int DEFAULT_PLAYLIST_SIZE        = 50;
        public final static int MAX_NEWS_ITEMS                       = 0;
```

```
        public final static int MAX_ADS                                    = 20;
        public final static int MAX_TIPS_ITEMS                             = 0;
        public final static int REFRESH_AT_SONGS_LEFT                      = 8;
        public final static int REFRESH_AT_NEW_RATINGS_COUNT   = 15;
        public final static int AD_THRESHOLD                = 30;
    public final static int NEWS_THRESHOLD              = 99999999;
    public final static int TIP_THRESHOLD               = 99999999;
        public final static byte ITEM_TYPE_SONG             = 1;
        public final static byte ITEM_TYPE_ALBUM             = 2;
        public final static byte ITEM_TYPE_ARTIST            = 3;
        // the size of the ratings cache FOR EACH user
        public final static int RATINGS_CACHE_INITIAL_SIZE     = 2000;
        public final static int RATING_UPDATE_LIST_INITIAL_SIZE = 100;
        // for updating the ratings caches
        public static final int PROPAGATE_DIRTY_RATING_SLEEP_TIME = 60 * 1000; // every 60 seconds
        public static final String POST_HEADER = "POST /servlet/playlist HTTP/1.0";
        public static final int PORT_NUMBER = 80;
}
```

Constants.java Page 2 of 2 11/05/99 1:24 PM

DBConnection

```
package com.launch.PlaylistGenerator;
import java.util.Properties;
import com.inet.tds.TdsDriver;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Connection;
import java.sql.Driver;
import java.sql.DriverManager;
import java.util.Date;
public class DBConnection
{
	private Connection conn;

	public static Driver DBDriver;

	public DBConnection() throws DBException
	{

		if (DBConnection.DBDriver == null)
			DBConnection.initializeDriver();

		if (DBConnection.DBDriver == null)
			return;

		String url = "jdbc:inetdae:"
			+ Constants.DB_SERVER
			+ ":"
			+ Constants.DB_PORT
			+ "?sql7=true&database="
			+ Constants.DB_DBNAME
			+ "&user="
			+ Constants.DB_USERNAME
			+ "&password="
			+ Constants.DB_PASSWORD
			+ "";

		try
		{
			conn = DBConnection.DBDriver.connect(url, null);
		}
		catch (SQLException oops)
		{
			throw new DBException(oops);
		}
		catch (Exception err)
		{
			Util.debug("Exception: " + err.toString());
		}
	}

	private static void initializeDriver()
	{
		DBDriver = new com.inet.tds.TdsDriver();

	}

	private DBResultSet execute(String sql, boolean printSQL) throws DBException
	{
		if (printSQL)
```

```
                Util.debug(Util.newLine + Thread.currentThread().getName() + " Running SQL: " + sql);
        DBResultSet myRs = new DBResultSet();
        try
        {

                // if we don't have a query, don't run it. It'll hang
                if (sql.length() <= 0)
                        return myRs;

                Statement query = conn.createStatement();

                if (query.execute(sql))
                {
                        myRs.setResultSet(query.getResultSet());
                }
        }
        catch (SQLException oops)
        {
                System.err.println(Util.newLine + (new Date()).toString() + " DBException: " +
Thread.currentThread().getName() + " Running SQL: " + sql + ", exception: " + oops.toString());
                oops.printStackTrace();
                throw new DBException(oops);
        }

        return myRs;
    }

    public void executeUpdate(String sql, boolean printSQL) throws DBException
    {
        if (printSQL)
                Util.debug(Util.newLine + Thread.currentThread().getName() + " Running SQL: " + sql);
        try
        {

                // if we don't have a query, don't run it. It'll hang
                if (sql.length() <= 0)
                        return;

                Statement query = conn.createStatement();

                query.executeUpdate(sql);
        }
        catch (SQLException oops)
        {
                // when we call a stored proc that gets a text pointer this happens,
                // so ignore it
                if (oops.getMessage().indexOf("Unknown datatype") > -1)
                {
    //                  System.err.println("ignoring unknown datatype exception");
                        return;
                }

                System.err.println(Util.newLine + (new Date()).toString() + " DBException: " +
Thread.currentThread().getName() + " Running SQL: " + sql + ", exception: " + oops.toString());
                oops.printStackTrace();
                throw new DBException(oops);
        }

    }

    public DBResultSet executeSQL(String sql) throws DBException
```

```
        {
                return execute(sql, true);
        }

        public DBResultSet executeSQL(String sql, boolean printSQL) throws DBException
        {
                return execute(sql, printSQL);
        }

        public DBPreparedStatement prepareStatement(String sql) throws DBException
        {

                try
                {
                        return new DBPreparedStatement(conn.prepareStatement(sql));
                }
                catch (SQLException oops)
                {
                        System.err.println(Util.newLine + (new Date()).toString() + " DBException in
prepareStatement: " + Thread.currentThread().getName() + ", exception: " + oops.toString());
                        oops.printStackTrace();
                        throw new DBException(oops);
                }

        }

        public boolean close() throws DBException
        {
                if (conn == null)
                        return false;

                try
                {
                        conn.close();
                        conn = null;
                        return true;
                }
                catch (SQLException oops)
                {
                        throw new DBException(oops);
                }
        }

        public void finalize() throws DBException
        {
                // in case someone forgets
                close();
        }

}
```

DBException

```
package com.launch.PlaylistGenerator;
import java.sql.SQLException;
public class DBException extends Exception
{
        SQLException oops;

        public DBException(SQLException oops)
        {
                this.oops = oops;
        }

        public String getMessage()
        {
                return oops.toString();
        }
}
```

DBException.java Page 1 of 1 11/05/99 1:26 PM

DBPreparedStatement

```
package com.launch.PlaylistGenerator;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Date;
public class DBPreparedStatement
{
	PreparedStatement statement;

	public DBPreparedStatement(PreparedStatement statement)
	{
		this.statement = statement;
	}

	public void setBytes(int parameterIndex, byte x[]) throws DBException
	{
		try
		{
			if (statement != null)
			{
				statement.setBytes(parameterIndex, x);
			}
		}
		catch (SQLException e)
		{
			throw new DBException(e);
		}
	}

	public void executeUpdate() throws DBException
	{
		Util.debug(Util.newLine + Thread.currentThread().getName() + " Running prepared statement");

		if (statement == null)
			return;

		try
		{
			statement.executeUpdate();

		}
		catch (SQLException oops)
		{
			System.err.println(Util.newLine + (new Date()).toString() + " DBException: " +
Thread.currentThread().getName() + " Running Statement, exception: " + oops.toString());
			oops.printStackTrace();
			throw new DBException(oops);
		}
	}

}
```

DBPreparedStatement.java Page 1 of 1 11/05/99 1:32 PM

DBResultSet

```
package com.launch.PlaylistGenerator;
import java.util.Date;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.io.InputStream;
public class DBResultSet
{
        private ResultSet rs;
        private boolean atEOF = false;
        private boolean atBOF = true;
        public void setResultSet(ResultSet aRS) throws DBException
        {
                try
                {
                        rs = aRS;
                        if (rs != null)
                                atBOF = !rs.next();
                }
                catch (SQLException oops)
                {
                        throw new DBException(oops);
                }
        }
        public int getInt(String columnName) throws DBException
        {
                try
                {
                        return rs.getInt(columnName);
                }
                catch (SQLException oops)
                {
                        throw new DBException(oops);
                }
        }
        public int getInt(int position) throws DBException
        {
                try
                {
                        return rs.getInt(position);
                }
                catch (SQLException oops)
                {
                        throw new DBException(oops);
                }
        }

        public InputStream getAsciiStream(String columnName) throws DBException
        {
                try
                {
                        return rs.getAsciiStream(columnName);
                }
                catch (SQLException oops)
                {
                        throw new DBException(oops);
                }
        }
```

```
public short getShort(String columnName) throws DBException
{
        try
        {
                return rs.getShort(columnName);
        }
        catch (SQLException oops)
        {
                throw new DBException(oops);
        }
}
public boolean getBoolean(String columnName) throws DBException
{
        try
        {
                return rs.getBoolean(columnName);
        }
        catch (SQLException oops)
        {
                throw new DBException(oops);
        }
}

public byte[] getBytes(String columnName) throws DBException
{
        try
        {
                return rs.getBytes(columnName);
        }
        catch (SQLException oops)
        {
                throw new DBException(oops);
        }
}

public float getFloat(String columnName) throws DBException
{
        try
        {
                return rs.getFloat(columnName);
        }
        catch (SQLException oops)
        {
                throw new DBException(oops);
        }
}
public float getFloat(int position) throws DBException
{
        try
        {
                return rs.getFloat(position);
        }
        catch (SQLException oops)
        {
                throw new DBException(oops);
        }
}
public String getString(String columnName) throws DBException
{
        try
        {
                return rs.getString(columnName);
```

```
                }
                catch (SQLException oops)
                {
                        throw new DBException(oops);
                }
        }
        public Date getDate(String columnName) throws DBException
        {
                try
                {
                        return rs.getDate(columnName);
                }
                catch (SQLException oops)
                {
                        throw new DBException(oops);
                }
        }
        public Timestamp getTimestamp(String columnName) throws DBException
        {
                try
                {
                        return rs.getTimestamp(columnName);
                }
                catch (SQLException oops)
                {
                        throw new DBException(oops);
                }
        }
        public boolean getBOF() throws DBException
        {
                return atBOF;
        }
        public boolean getEOF() throws DBException
        {
                return atEOF;
        }
        public void next() throws DBException
        {
                try
                {
                        atEOF = !rs.next();
                }
                catch (SQLException oops)
                {
                        throw new DBException(oops);
                }
        }
        public boolean wasNull() throws DBException
        {
                try
                {
                        return rs.wasNull();
                }
                catch (SQLException oops)
                {
                        throw new DBException(oops);
                }
        }
}
```

DJ

```
package com.launch.PlaylistGenerator;
public class DJ
{
        public int userID;
        public String alias;
        public DJ (int id, String name)
        {
                this(id);
                alias = name;
        }

        public DJ (int id)
        {
                userID = id;
        }

}
```

DJ.java Page 1 of 1 11/05/99 1:26 PM

DJList

```
package com.launch.PlaylistGenerator;
import java.util.Vector;
public class DJList extends Vector
{

        public DJ djAt(int i)
        {
                return (DJ) elementAt(i);
        }

        public String inList()
        {
                Integer list[] = new Integer[size()];

                int last = 0;

                for (int i = 0; i < this.size(); i++)
                {
                        list[i] = new Integer(djAt(i).userID);
                }

                return Util.join(", ", list);
        }
        public boolean load(DBConnection conn, int userID, int moodID)
        {

                short djCount = 0;

                try
                {

                        DBResultSet rs = conn.executeSQL("exec sp_lcoGetDJs_xsxx "
                                                                        + userID + ", "
                                                                        + moodID);

                        while (!rs.getBOF() && !rs.getEOF())
                        {

                                addElement(new DJ(rs.getInt("djID")));

                                rs.next();
                                djCount++;
                        }

                        Util.debug(Thread.currentThread().getName() + " added " + djCount + " DJs");
                }
                catch (DBException oops)
                {
                        Util.debug("DB Exception in DJList::load: " + oops.getMessage());
                }

                return (djCount > 0);
        }

        public Vector asIDVector()
        {
                Vector users = new Vector(10);

                for (int i = 0; i < this.size(); i++)
```

```
        {
                users.addElement(new Integer(((DJ) elementAt(i)).userID));
        }
65      return users;
    }
}
```

DJList.java Page 2 of 2 11/05/99 1:28 PM

FrequencyCounter

```
package com.launch.PlaylistGenerator;
import java.util.*;
/**
 * FrequencyCounter is a Hashtable of the form (Object, Integer)
 * <br><br>
 * okay I realize the getLargest and getSmallestValue
 * methods are very inefficient (CPU wise) but these methods
 * aren't called often, if they are then some one should
 * do an nlog(n) sort on them then just pick out the largest
 * after that
 **/
public class FrequencyCounter extends Hashtable
{
        public FrequencyCounter()
        {
        }

        public FrequencyCounter(int i)
        {
                super(i);
        }

        public void incrementValue(Object o)
        {
                Integer i=(Integer)get(o);

                if (i==null)
                {
                        put(o, new Integer(1));
                }
                else
                {
                        put(o, new Integer((i.intValue())+1));
                }
        }


        public FrequencyCounter getLargest(int n)
        {
                FrequencyCounter fc=new FrequencyCounter(n+10);

                Integer temp_int;
                Object temp_object;
                Object smallest_value_key=null;
                int smallest_value;
                Enumeration e=keys();

                while (e.hasMoreElements())
                {
                        temp_object=e.nextElement();
                        temp_int=(Integer)get(temp_object);

                        if (fc.size()>=n)
                        {
                                smallest_value_key=fc.getSmallestValue();
                                smallest_value=((Integer)fc.get(smallest_value_key)).intValue();
                                if (temp_int.intValue()>smallest_value)
                                {
                                        fc.remove(smallest_value_key);
```

```
                                        fc.put(temp_object, temp_int);
                                }
                        }
                        else
                        {
                                fc.put(temp_object, temp_int);
                        }
                }

                return(fc);
        }


        /** @return null if list is empty */
        public Object getSmallestValue()
        {
                int smallest_value=Integer.MAX_VALUE;
                Object smallest_value_key=null;

                int temp_int;
                Object temp_object;

                Enumeration e=keys();
                while(e.hasMoreElements())
                {
                        temp_object=e.nextElement();
                        temp_int=((Integer)get(temp_object)).intValue();

                        if (temp_int<smallest_value)
                        {
                                smallest_value=temp_int;
                                smallest_value_key=temp_object;
                        }
                }

                return(smallest_value_key);
        }


        //*********************************************************
        // The following is a test function

        public static void main(String argv[])
        {
                FrequencyCounter fc=new FrequencyCounter();

                fc.incrementValue("one");
                fc.incrementValue("two");
                fc.incrementValue("two");

                fc.incrementValue("three");
                fc.incrementValue("three");
                fc.incrementValue("three");

                fc.incrementValue("four");
                fc.incrementValue("four");
                fc.incrementValue("four");
                fc.incrementValue("four");
                System.out.println(fc);
                System.out.println("smallest "+ fc.getSmallestValue());
                System.out.println("largest 2" + fc.getLargest(2));
        }
```

```
}
```

GeneratorParameters

```
package com.launch.PlaylistGenerator;

import javax.servlet.http.HttpServletRequest;
public class GeneratorParameters
{

        private int userID, moodID, djID;
        private Bandwidth speed;
        private boolean debug, matrix, forceRefresh, dontsave;
        private MediaFormat format;

        private boolean moodIDSet   = false;
        private boolean djIDSet     = false;

        private int debugFormat = Util.DISPLAY_TEXT;

        public Bandwidth speed()
        {
                return speed;
        }

        public MediaFormat format()
        {
                return format;
        }

        public int debugFormat()
        {
                return debugFormat;
        }

        public int userID()
        {
                return userID;
        }

        public int moodID()
        {
                return moodID;
        }

        public int djID()
        {
                if (djIDSet)
                        return djID;

                return userID;
        }

        public boolean debug()
        {
                return debug;
        }

        public boolean matrix()
        {
                return matrix;
        }
```

```
        public boolean forceRefresh()
        {
                return forceRefresh;
        }

        public boolean dontsave()
        {
                return dontsave;
        }

        public GeneratorParameters(HttpServletRequest request)
        {

                debug       = (request.getParameter("ralph")       != null);
                matrix      = (request.getParameter("matrix")      != null);
                forceRefresh = (request.getParameter("forceRefresh") != null);
                dontsave    = (request.getParameter("dontsave")    != null);

                String debugFormatString = request.getParameter("format");

                if (debugFormatString != null && debugFormatString.equals("html"))
                        debugFormat = Util.DISPLAY_HTML;

                try { userID  = Integer.parseInt(request.getParameter("u")); }
                catch (NumberFormatException e) { userID = 0; }

                try { moodID = Integer.parseInt(request.getParameter("m")); }
                catch (NumberFormatException e) { moodID = 0; moodIDSet = false;}
                moodIDSet = true;

                try { djID = Integer.parseInt(request.getParameter("d")); }
                catch (NumberFormatException e) { djID = userID; djIDSet = false;}

                djIDSet = true;

                if (djID <= 0)
                {
                        djID = userID;
                        djIDSet = false;
                }

                speed = new Bandwidth(request.getParameter("b"));

                format = new MediaFormat();

        }

}
```

GenreIndex

```
package com.launch.PlaylistGenerator;
import java.util.Hashtable;
import java.util.Vector;
public class GenreIndex extends Hashtable
{

        public GenreIndex(int x, int y)
        {
                super(x, y);
        }

        public void add(short index, SongInfo info)
        {
                SongList list = get(index);

                if (list == null)
                {
                        list = new SongList();
                        put(new Short(index), list);
                }

                list.addElement(info);

        }

        public SongList get(int index)
        {
                return (SongList) get(new Short((short) index));
        }

        public int countInGenreList(GenreList myGenres)
        {
                int result = 0;

                SongList list;

                for (int i = 0; i < myGenres.size(); i++)
                {
                        list = get(myGenres.genreAt(i));

                        if (list != null)
                        {
                                result += list.size();
                        }
                }
                return result;
        }

        /**
         * returns a COPY of the list of songs in genres
         */
        public SongList getInGenreList(GenreList myGenres)
        {
                SongList result = new SongList();

                for (int i = 0; i < myGenres.size(); i++)
                {
                        result.addElements(get(myGenres.genreAt(i)));
                }
```

```
                return result;
        }

        /**
         * returns a COPY of the list of songs in a genre
         */
        public SongList getInGenre(int genreID)
        {
                SongList list = get(genreID);
                SongList result;

                if (list == null)
                        list = new SongList();
                result = (SongList) list.clone();

                return result;
        }

}
```

GenreIndex.java Page 2 of 2 11/05/99 1:28 PM

GenreList

```
package com.launch.PlaylistGenerator;
import java.util.Hashtable;
public class GenreList
{
	private int genres[];
	private Hashtable hash;

	private byte next;

	public boolean allGenres = true;

	public GenreList()
	{
		hash = new Hashtable(1,1);
		genres = new int[100];
	}

	public int add(short genreID)
	{
		allGenres = false;
		hash.put(new Short(genreID), new Boolean(true));
		genres[next] = genreID;
		next++;

		return genres[next - 1];
	}

	public int size()
	{
		return next;
	}

	public int genreAt(int pos)
	{
		return genres[pos];
	}

	public boolean exists(Short genreID)
	{

		if (next == 0)
			return true;

		else
			return hash.containsKey(genreID);
	}

	public String toString() {

		String result = "";

		for (int i = 0; i < size(); i++)
		{
			result = result.concat(genreAt(i) + ", ");

		}

		return result;
	}
```

}
GenreList.java Page 2 of 2 11/05/99 1:26 PM

GetAds

```
package com.launch.PlaylistGenerator;
import java.util.Date;
import java.util.Vector;
public class GetAds extends Thread
{
        Vector ads;
        int userID;
        short mediaType;

        public GetAds(Vector ads, int userID, short mediaType)
        {
                this.ads       = ads;
                this.userID    = userID;
                this.mediaType = mediaType;
        }
        public void run()
        {
                Date startDate = new Date();
                Thread.currentThread().setName("GetAds");

                int rowCount = 0;
                int count    = 0;

                Clip aClip;
                int clipID, mediaID;
                Date lastPlayed;
                String clipName;


                String sql = new String("exec sp_lcGetAds_xsxx "
                                                                        + userID
                                                                + ", "
                                                                + mediaType
                                                                );

                try
                {
                        DBConnection conn = new DBConnection();
                        DBResultSet rs = conn.executeSQL(sql);
                        while (!rs.getBOF() && !rs.getEOF() && count < Constants.MAX_ADS)
                        {
                                ads.addElement(new Clip(rs.getInt("clipID"),
                                                                        Clip.TYPE_AD,
                                                                        rs.getInt("mediaID"),
                                                                        rs.getString("clipName"),
                                                                        rs.getDate("lastPlayed")));
                                count++;
                                rs.next();
                                rowCount++;
                        }

                        conn.close();
                }
                catch (DBException oops)
                {
                        Util.debug("DB Exception: " + oops.getMessage());
                }
                Util.debug(Thread.currentThread().getName() + " added " + count + " ads");
                Util.printElapsedTime(Thread.currentThread().getName(), startDate);
```

```
    }
}
```

GetAds.java Page 2 of 2 11/05/99 1:37 PM

GetBDSStations

```
package com.launch.PlaylistGenerator;
import java.util.Date;
public class GetBDSStations extends Thread
{

        int userID;
        int moodID;
        StationList stations;

        public GetBDSStations(int userID, int moodID, StationList stations)
        {
                this.userID  = userID;
                this.moodID  = moodID;
                this.stations = stations;
        }
        public void run()
        {
                Date startDate = new Date();
                Thread.currentThread().setName("GetBDSStations");

                int rowCount = 0;

                String sql = "sp_lcGetBDSNames_xsxx " + userID + ", " + moodID;

                try
                {

                        DBConnection conn = new DBConnection();

                        DBResultSet rs = conn.executeSQL(sql);
                        while (!rs.getBOF() && !rs.getEOF())
                        {
                                int bdsID = rs.getInt("bdsID");
                                stations.addElement(new Station(bdsID));
                                rowCount++;
                                rs.next();
                        }

                        conn.close();

                }
                catch (DBException oops)
                {
                        Util.debug("DB Exception in GetBDSStations: " + oops.getMessage());
                }

                Util.debug(Thread.currentThread().getName() + " got " + rowCount + " BDS station
subscriptions");
                Util.printElapsedTime(Thread.currentThread().getName(), startDate);
        }
}
```

GetGenres

```
package com.launch.PlaylistGenerator;
import java.util.Date;
public class GetGenres extends Thread
{

        GenreList genres;
        int djID;
        int moodID;

        public GetGenres(GenreList genres, int djID, int moodID)
        {
                this.genres = genres;
                this.moodID = moodID;
                this.djID = djID;

        }
        public void run()
        {
                Date startDate = new Date();
                Thread.currentThread().setName("GetGenres");

                int rowCount = 0;

                try
                {
                        DBConnection conn = new DBConnection();

                        DBResultSet rs = conn.executeSQL("exec sp_lcGetGenreNamesForUser_xsxx "
                                                                + djID + ", "
                                                                + moodID);

                        while (!rs.getBOF() && !rs.getEOF())
                        {
                                genres.add((short) rs.getInt("genreID"));
                                rowCount++;
                                rs.next();
                        }

                        conn.close();
                }
                catch (DBException oops)
                {
                        Util.debug("DB Exception: " + oops.getMessage());
                }

                Util.debug(Thread.currentThread().getName() + " added " + rowCount + " genres");
                Util.printElapsedTime(Thread.currentThread().getName(), startDate);
        }
}
```

GetItemRatingsFromDB

```
package com.launch.PlaylistGenerator;
import java.util.*;
public final class GetItemRatingsFromDB extends Thread
{
        private Vector userIDs;
        private Vector results;
    //-------------------------------------------------------
        public GetItemRatingsFromDB(Vector userIDs, Vector results)
        {
                this.userIDs = userIDs;
                this.results = results;
        }
        public void run()
        {
                Thread.currentThread().setName("GetItemRatingsFromDB");
                Util.debug(Thread.currentThread().getName() + " thread started");
                Date startDate = new Date();

                try
                {
                        String sql = "SELECT iUserID_FK userID, iSourceTableID_L type,
iItemID_FK itemID, tiRating rating FROM a125ItemRating WHERE iUserID_FK IN (" +
RatingsCache.GetVectorAsCommaDelimitedList(userIDs) + ')';
                        DBConnection conn = new DBConnection();
                        DBResultSet rs = conn.executeSQL(sql);
                        CachedRating cr;

                        byte type;
                        while (!rs.getBOF() && !rs.getEOF())
                        {
                                cr = new CachedRating(rs.getInt("userID"), rs.getInt("itemID"), (byte)
rs.getInt("rating"), sourceTableIDToType(rs.getInt("type")));
                                results.addElement(cr);
                                rs.next();
                        }
                        conn.close();
                }
                catch (DBException oops)
                {
                        System.err.println("DBException in GetItemRatingsFromDB: " +
oops.getMessage());
                }
                Util.printElapsedTime(Thread.currentThread().getName(), startDate);
        }

        public final static byte sourceTableIDToType (int type)
        {
                if (type == 260)
                        return Constants.ITEM_TYPE_ARTIST;

                // assume album (243)

                return Constants.ITEM_TYPE_ALBUM;
        }

}
```

GetLastPlayed

```
package com.launch.PlaylistGenerator;
import java.util.Date;
import java.text.DateFormat;
import javax.servlet.ServletOutputStream;
public class GetLastPlayed extends Thread
{
        PlayDates lastPlayed;
        int userID;
        ServletOutputStream out;

        public GetLastPlayed(PlayDates lastPlayed, int userID, ServletOutputStream out)
        {
                this.lastPlayed = lastPlayed;
                this.userID     = userID;
                this.out        = out;
        }
        public void run()
        {
                Date startDate = new Date();
                Thread.currentThread().setName("GetLastPlayed");

                // returns: songID, lastPlayed

                try
                {
                        DBConnection conn = new DBConnection();

                        Util.printElapsedTime(Thread.currentThread().getName() + " got a dbConnection",
startDate);
                        lastPlayed.load(conn, userID);

                        Util.printElapsedTime(Thread.currentThread().getName() + " loaded dates", startDate);
                        // this is somewhat expensive, so only do it every so often

                        if (Util.random(10) == 1)
                        {
                                Util.debug("resaving lastPlayed for user " + userID);
                                lastPlayed.save(conn);
                        }

                        conn.close();
                }
                catch (DBException oops)
                {
                        Util.debug("DB Exception: " + oops.getMessage());
                }

                Util.out(out, Thread.currentThread().getName() + " loaded " + lastPlayed.size() + " dates");
                Util.printElapsedTime(Thread.currentThread().getName() + "done GetLastPlayed", startDate);
        }
}
```

GetNews

```
package com.launch.PlaylistGenerator;
import java.util.Date;
import java.util.Vector;
public class GetNews extends Thread
{
	Vector news;
	int userID;
	short mediaType;
	int moodID;

	public GetNews(Vector news, int userID, short mediaType, int moodID)
	{
		this.news = news;
		this.userID = userID;
		this.mediaType = mediaType;
		this.moodID = moodID;
	}
	public void run()
	{
		Date startDate = new Date();
		Thread.currentThread().setName("GetNews");

		int rowCount = 0;
		int count = 0;

		Clip aClip;
		int clipID, mediaID;
		Date lastPlayed;
		String clipName;

		/*
		sp_lcGetNews_xsxx	@userID int, @moodID int, @mediaType int
		returns clipID, clipName, mediaID, lastPlayed

		*/

		String sql = new String("exec sp_lcGetNews_xsxx "
									+ userID
		+ ","
		+ moodID
		+ ","
		+ mediaType
		);

		try
		{
			DBConnection conn = new DBConnection();
			DBResultSet rs = conn.executeSQL(sql);
			while(!rs.getBOF() && !rs.getEOF() && count < Constants.MAX_NEWS_ITEMS)
			{
				news.addElement(new Clip(rs.getInt("clipID"),
					Clip.TYPE_NEWS,
					rs.getInt("mediaID"),
					rs.getString("clipName"),
					rs.getDate("lastPlayed")));
				count++;
				rs.next();
				rowCount++;
			}
```

```
                    conn.close();
                }
                catch (DBException oops)
                {
                    Util.debug("DB Exception: " + oops.getMessage());
                }
                Util.debug(Thread.currentThread().getName() + " added " + count + " news items");
                Util.printElapsedTime(Thread.currentThread().getName(), startDate);
        }

}
```

GetNews.java Page 2 of 2 11/05/99 1:38 PM

GetPlaylist

```
package com.launch.PlaylistGenerator;
import java.util.Date;
public class GetPlaylist extends Thread
{
        Population songs;
        int userID;
        SongInfoCache cache;

        public GetPlaylist(Population songs, int userID, SongInfoCache cache)
        {
                this.songs = songs;
                this.userID = userID;
                this.cache = cache;
        }
        public void run()
        {
                Date startDate = new Date();
                Thread.currentThread().setName("GetPlaylist");

                SongInfo info = null;
                SimpleClip clip;
                int songID;
                int rowCount = 0;

                try
                {
                        DBConnection conn = new DBConnection();
                        Util.printElapsedTime(Thread.currentThread().getName() + " got a dbConnection",
startDate);
                        SimplePlaylist playlist = SimplePlaylist.load(conn, userID);
                        if (playlist != null)
                        {
                                for (int i = 0; i < playlist.songs.size(); i++)
                                {
                                        clip  = (SimpleClip) playlist.songs.elementAt(i);
                                        songID = clip.ID;

                                        songs.initSong(songID, Song.EXCLUDED);
                                        info = (SongInfo) cache.get(songID, SongInfoCache.TYPE_SONG);

                                        songs.artistCounts.increment(info.album.artist.ID);
                                        songs.albumCounts.increment(info.album.ID);

                                        rowCount++;
                                }
                        }

                        conn.close();
                }
                catch (DBException oops)
                {
                        Util.debug("DB Exception: " + oops.getMessage());
                }
                Util.debug(Thread.currentThread().getName() + " excluded " + rowCount + " songs");
                Util.printElapsedTime(Thread.currentThread().getName(), startDate);
        }

}
```

GetPlaylistServers

```
package com.launch.PlaylistGenerator;
import java.util.*;
/**
 **/
public final class GetPlaylistServers extends Thread
{
        public static int SLEEP_TIME = (3600*1000); // every hour
        public static int EXPECTED_SERVER_COUNT = 10;
        private GetPlaylistServersInterface personToNotify;
    //-------------------------------------------------------
        /**
         * @param personToNotify must not be null.
         **/
        public GetPlaylistServers(GetPlaylistServersInterface personToNotify)
        {
                this.personToNotify=personToNotify;
        }
        public void run()
        {
                Thread.currentThread().setName("getPlaylistServers");
                Util.debug(Thread.currentThread().getName() + " thread started");
                DBConnection conn;
                DBResultSet rs;
                Vector v;
                Date benchmark_date;
                try
                {
                        while (personToNotify!=null)
                        {
                                benchmark_date=new Date();
                                v=new Vector(EXPECTED_SERVER_COUNT);
                                conn = new DBConnection();
                                rs = conn.executeSQL("exec sp_lcGetRatingsCacheServers_xsxd");
                                while (!rs.getBOF() && !rs.getEOF())
                                {
                                        v.addElement(rs.getString("server"));
                                        rs.next();
                                }
                                conn.close();
                                personToNotify.updatePlaylistServers(v);
                                Util.printElapsedTime(Thread.currentThread().getName() + ", get " +
v.size() + " rows", benchmark_date);
                                Thread.sleep(SLEEP_TIME);
                        }
                }
                catch (Exception e)
                {
                        System.err.println(new Date().toString() + " Fatal Exception in
GetPlaylistServers:" + e.toString());
                }
                Util.debug(Thread.currentThread().getName() + " thread done");
        }
}
GetPlaylistServers.java  Page 2 of 2  11/05/99 1:37 PM
```

GetPlaylistServersInterface

```
package com.launch.PlaylistGenerator;
import java.util.*;
public interface GetPlaylistServersInterface
{
        /**
         * @param playlistServers will be a vector of strings, each string is an ip address of the form
xxx.xxx.xxx.xxx
         **/
        public void updatePlaylistServers(Vector playlistServers);
}
```

GetPlaylistServersInterface.java Page 1 of 1 11/05/99 1:28 PM

GetPopular

```
package com.launch.PlaylistGenerator;
import java.util.Date;
public class GetPopular extends Thread
{
        Population songs;
        SongList list;

        public GetPopular(Population songs, SongList list)
        {
                this.songs    = songs;
                this.list     = list;
        }
        public void run()
        {
                Date startDate = new Date();
                Thread.currentThread().setName("GetPopular");
                Song ditty;
                SongData data;
                SongInfo info;

                int rowCount = 0;

                if (list != null)
                {
                        for (int i = 0; i < list.size(); i++)
                        {

                                info = list.elementAt(i);

                                data = songs.getSongData(info.songID);

                                if (data != null)
                                {
                                        // we can't add it, but let's append the info while we're here

                                        data.setInfo(info);
                                }
                                else
                                {
                                        data = songs.initSongGetData(info.songID, Song.UNRATED);

                                        if (data != null)
                                        {
                                                data.querySource = data.SOURCE_POPULAR;
                                                data.setInfo(info);
                                        }
                                        rowCount++;
                                }
                        }

                }
                Util.debug(Thread.currentThread().getName() + " added " + rowCount + " songs");
                Util.printElapsedTime(Thread.currentThread().getName(), startDate);
        }

}
```

GetRatings

```
package com.launch.PlaylistGenerator;
import java.util.Date;
import java.util.Vector;
import java.util.Enumeration;
import javax.servlet.ServletOutputStream;
public class GetRatings extends Thread
{
        ItemsProfile profile;
        int userID;
        DJList djs;
        Population songs;
        SongInfoCache cache;
        ServletOutputStream out;

        public GetRatings(Population songs, ItemsProfile profile, int userID, DJList djs, SongInfoCache cache,
ServletOutputStream out)
        {
                this.profile = profile;
                this.userID  = userID;
                this.djs     = djs;
                this.cache   = cache;
                this.songs   = songs;
        }
        public void run()
        {
                Date startDate = new Date();
                Thread.currentThread().setName("GetRatings");

                int rowCount = 0;

                // make a users vector from the users and djs

                Vector users = djs.asIDVector();
                users.addElement(new Integer(userID));
                Util.out(out, "GetRatings getting ratings for users " + users.toString());

                Vector ratings = cache.ratingsCache.getRatings(users);

                Util.printElapsedTime("GetRatings after all ratings retreived", startDate);

                CachedRating cached;
                int djID, itemID;
                byte rating, type;
                SongData data;
                short songType = Song.EXPLICIT;
                SongInfo info;
                int artistID;
                Item theItem;

                int songRatings = 0;
                int itemRatings = 0;

                int userSongRatings = 0;
                int userItemRatings = 0;
                int djSongRatings = 0;
                int djItemRatings = 0;

                for (Enumeration e = ratings.elements(); e.hasMoreElements() ;)
                {
```

```
                    cached = (CachedRating) e.nextElement();

                    djID   = cached.userID;
                    itemID = cached.itemID;
                    rating = cached.rating;
                    type   = cached.type;
                    // 0 is not a valid userId
                    // ratings < 0 mean it was unrated
                    if (djID != 0 || rating < 0)
                    {
                        if (type == Constants.ITEM_TYPE_SONG)
                        {
                            songRatings++;

                            // store the user's rating
                            if (userID == djID)
                            {
                                userSongRatings++;

                                if (rating == 0)
                                {
                                    songs.initSong(itemID, Song.EXCLUDED);
                                    info = (SongInfo) cache.get(itemID,
SongInfoCache.TYPE_SONG);
                                    addToAverage(info, 0);
                                }
                                else
                                {

                                    data = songs.initSongGetData(itemID, songType);
                                    if (data != null)
                                    {
                                        info = (SongInfo) cache.get(itemID,
SongInfoCache.TYPE_SONG);

                                        // if the song isn't in the cache, it's not
encoded
                                        // and we can't play it
                                        if (info == null)
                                        {
                                            songs.initSong(itemID,
Song.EXCLUDED);
                                        }
                                        else
                                        {
                                            data.setInfo(info);
                                            data.querySource =
SongData.SOURCE_RATED;
                                            data.rating.set(rating,
SongRating.RATING_SOURCE_EXPLICIT);

                                            // add this rating to all ratings by
this user for the artist
                                            addToAverage(info, rating);
                                        }
                                    }
                                }
                            }
                            else // this is another user's song rating
```

```
                                        {
                                                djSongRatings++;
                                                data = songs.initSongGetData(itemID, Song.UNRATED);
                                                if (data != null)
                                                {
                                                        data.querySource = SongData.SOURCE_DJS;
                                                        data.djsAverage.add(rating);
                                                }
                                        }
                                }
                                // don't count various artists ratings
                                else if (!(type == Constants.ITEM_TYPE_ARTIST &&
ArtistInfo.isVariousArtists(itemID)))
                                {
                                        itemRatings++;
                                        theItem = profile.put(itemID);
                                        if (djID == userID)
                                        {
                                                userItemRatings++;
                                                theItem.userRating.set(rating);
                                        }
                                        else
                                        {
                                                djItemRatings++;
                                                theItem.djsAverage.add(rating);
                                        }
                                }
                        }
                        rowCount++;
                }
                Util.out(out, Thread.currentThread().getName() + " added "
                                        + songRatings + " song ratings ("
                                        + userSongRatings + " user, "
                                        + djSongRatings + " dj) "
                                        + "and " + itemRatings + " item ratings ("
                                        + userItemRatings + " user, "
                                        + djItemRatings + " dj)"
                                );
                Util.printElapsedTime(Thread.currentThread().getName(), startDate);
        }

        private void addToAverage(SongInfo info, int rating)
        {
                if (info != null)
                {
                        (profile.put(info.album.artist.ID)).songAverage.add(rating);
                }
        }

        private String userCriteria()
        {
```

```
                if (djs.size() <= 0)
                        return " = " + userID;

                return "IN (" + userID + ", " + djs.inList() + ")";
        }
}
```

GetRatings.java Page 4 of 4 11/05/99 1:35 PM

GetRatingsCacheUsers

```
package com.launch.PlaylistGenerator;
import java.util.*;
import java.net.*;
/**
 **/
public final class GetRatingsCacheUsers extends Thread
{
		private static int SLEEP_TIME = (10 * 60 * 1000); // update every 10 minutes
		private static int EXPECTED_TOP_USER_SIZE = 100;
		private GetRatingsCacheUsersInterface personToNotify;
		private static final int UPDATE_DB_CACHED_USERS_SLEEP_COUNT = 6 * 8; // three times
every day (6*8*SLEEP_TIME)
	//---------------------------------------------------------
		/**
		 * @param personToNotify must not be null.
		 **/
		public GetRatingsCacheUsers(GetRatingsCacheUsersInterface personToNotify)
		{
			this.personToNotify = personToNotify;
		}
		public void run()
		{
			Thread.currentThread().setName("GetRatingsCacheUsers");
			Util.debug(Thread.currentThread().getName() + " thread started");
			DBConnection conn;
			String myIP;
			DBResultSet rs;
			Vector v;
			Date benchmark_date;
			try
			{
				myIP = InetAddress.getLocalHost().getHostAddress();
				int update_db_users_list =
UPDATE_DB_CACHED_USERS_SLEEP_COUNT;
				while (personToNotify != null)
				{
					benchmark_date = new Date();
					v = new Vector(EXPECTED_TOP_USER_SIZE);
					conn = new DBConnection();
					rs = conn.executeSQL("exec sp_lcGetUsersToCache_isxd '" + myIP +
"'");
					while (!rs.getBOF() && !rs.getEOF())
					{
						v.addElement(new Integer(rs.getInt("userID")));
						rs.next();
					}
					personToNotify.updateCachedUsers(v);
					Util.printElapsedTime(Thread.currentThread().getName() + ", get " +
v.size() + " rows", benchmark_date);
					Thread.sleep(SLEEP_TIME);
					//---
					if (update_db_users_list <= 0)
					{
						// do the update

						Util.debug(new Date().toString() + " Updating
RatingsCacheUserList");

						try
```

```
                                        {
                                                Hashtable h =
personToNotify.getMostFrequentlyUsedUsers(EXPECTED_TOP_USER_SIZE);

                                                if (h != null && h.size() > 0)
                                                {
                                                        String the_command = "exec
sp_lcDeleteRatingsCacheUsers_xxxd";
                                                        conn.executeSQL(the_command);
                                                        Enumeration e = h.keys();
                                                        while (e.hasMoreElements())
                                                        {
                                                                the_command = "exec
sp_lcAddRatingsCacheUser_ixxx " + e.nextElement();
                                                                conn.executeSQL(the_command);
                                                        }
                                                }

                                                conn.close();
                                        }
                                        catch (DBException dbe)
                                        {
                                                System.err.println(new Date().toString() + "
DBException in GetRatingsCacheUsers: " + dbe.toString());
                                                dbe.printStackTrace();
                                        }

                                        update_db_users_list =
UPDATE_DB_CACHED_USERS_SLEEP_COUNT;
                                }
                                else
                                {
                                        Util.debug("update_db_users_list is " + update_db_users_list);
                                        update_db_users_list--;
                                }
                                //---

                                conn.close();
                        }
                }
                catch (Exception e)
                {
                        System.err.println(new Date().toString() + " Fatal Exception in
GetRatingsCacheUsers: " + e.getMessage());
                        e.printStackTrace();
                }
                Util.debug(Thread.currentThread().getName() + " thread done");
        }
}
GetRatingsCacheUsers.java       Page 2 of 3       11/05/99 1:23 PM
```

GetRatingsCacheUsersInterface

```
package com.launch.PlaylistGenerator;
import java.util.*;
public interface GetRatingsCacheUsersInterface
{
        /**
         * @param topUsers will be a vector of Integers, where each integer is a userID
         **/
        public void updateCachedUsers(Vector topUsers);

        /**
         * This method will return a hash of (Integer USERID, Intger Requests)
         * @param i is the number of users to get
         * @return null if no statistics
         **/
        public Hashtable getMostFrequentlyUsedUsers(int i);
}
```

GetRatingsCacheUsersInterface.java Page 1 of 1 11/05/99 1:28 PM

GetRecentlyPlayed

```
package com.launch.PlaylistGenerator;
import java.util.Date;
public class GetRecentlyPlayed extends Thread
{
        Population songs;
        int userID;

        public GetRecentlyPlayed(Population songs, int userID)
        {
                this.songs = songs;
                this.userID = userID;
        }
        public void run()
        {
                Date startDate = new Date();
                Thread.currentThread().setName("GetRecentlyPlayed");

                int rowCount = 0;

                String sql = new String("exec sp_lcGetRecentlyPlayedSongs_xsxx "
                                                                                + userID);
                int songID, albumID, artistID;

                try
                {
                        DBConnection conn = new DBConnection();
                        DBResultSet rs = conn.executeSQL(sql);
                        while(!rs.getBOF() && !rs.getEOF())
                        {

                                // returns songID, albumID, artistID, lastPlayed

                                albumID  = rs.getInt("albumID");
                                songID   = rs.getInt("songID");
                                artistID = rs.getInt("artistID");

                                // don't play these songs so soon again
                                songs.initSong(songID, Song.EXCLUDED);

                                songs.artistCounts.increment(artistID);
                                songs.albumCounts.increment(albumID);

                                rs.next();
                                rowCount++;
                        }

                        conn.close();

                }
                catch (DBException oops)
                {
                        Util.debug("DBException: " + oops.getMessage());
                }

                Util.debug(Thread.currentThread().getName() + " added " + rowCount + " songs");
                Util.printElapsedTime(Thread.currentThread().getName(), startDate);
        }
```

```
}
```

GetRecentlyPlayed.java Page 2 of 2 11/05/99 1:26 PM

GetSongInfoServlet

```
package com.launch.PlaylistGenerator;
import java.util.*;
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;
/**
*---------------------------------------------------------------
*
* GetSongInfoServlet
* @author Jeff Boulter
*---------------------------------------------------------------
*/
public class GetSongInfoServlet extends HttpServlet
{
        public static final byte ONLINE_TIMEOUT = 10;
        //----------------------
        /**
         * Handle requests...
         */
        public void doGet (
                HttpServletRequest      request,
                HttpServletResponse     response
                ) throws ServletException, IOException
        {
                String userID;
                String volume;
                String djID;
                String djName;
                String djPosessive;
                String songName = "";
                String albumName = "";
                String artistName = "";
                int songID = 0;
                int albumID = 0;
                int artistID = 0;
                int commRating = 0;
                Date dateAdded = new Date();
                byte origin = 0;
                int mediaID = 0;
                int year = 0;
                int songRating = -1;
                int albumRating = -1;
                int artistRating = -1;
                // get stream for output
                ServletOutputStream out;
                response.setContentType("text/html");
                out = response.getOutputStream();
                response.setHeader("Pragma", "no-cache");
                response.setHeader("Cache-control", "no-cache");
                response.setHeader("Expires", "0");
                try
                {
                        userID = request.getParameter("rater");
                        if (userID == null)
                        {
                                out.println("no userID passed");
                                return;
                        }
```

```
                        DBConnection conn = new DBConnection();
                        djID  = request.getParameter("djID");
                        djName = request.getParameter("djName");
                        if (djID == null || djID.equals(userID))
                        {
                                djName = "You";
                                djPosessive = "Your";
                        }
                        else
                        {
                                djPosessive = djName + "'s";
                        }
                        DBResultSet rs = conn.executeSQL("exec sp_lcGetPlayingInfoForUser_xsxx "
+ userID);
                        while (!rs.getBOF() && !rs.getEOF())
                        {
                                songName  = rs.getString("song");
                                albumName  = rs.getString("album");
                                artistName = rs.getString("artist");
                                songID    = rs.getInt("songID");
                                albumID   = rs.getInt("albumID");
                                artistID  = rs.getInt("artistID");
                                commRating = rs.getInt("commRating");
                                if (commRating <= 0) { commRating = -1;}
                                origin    = (byte) rs.getInt("origin");
                                mediaID   = rs.getInt("mediaID");
                                year      = rs.getInt("year");
                                dateAdded  = rs.getTimestamp("dateAdded");
                                songRating  = rs.getInt("songRating");
                                albumRating  = rs.getInt("albumRating");
                                artistRating = rs.getInt("artistRating");
                                rs.next();
                        }
                        int exclusive = isExclusive(albumName);
                        int newStatus = isNew(dateAdded);
                        int popular  = isPopular(commRating);
                        String djs = "";

                        if (origin == SongData.SOURCE_DJS_ALBUM)
                                djs = djRatings(conn, userID, albumID,
Constants.ITEM_TYPE_ALBUM);
                        else if (origin == SongData.SOURCE_DJS_ARTIST)
                                djs = djRatings(conn, userID, artistID,
Constants.ITEM_TYPE_ARTIST);
                        else
                                djs = djRatings(conn, userID, songID,
Constants.ITEM_TYPE_SONG);

                        out.print(
                                        "media_id=" + mediaID + "&"
                                        + "song_id=" + songID + "&"
                                        + "song_name=" + escape(songName) + "&"
                                        + "album_id=" + albumID + "&"
                                        + "album_name=" + escape(albumName +
formatAlbumYear(year)) + "&"
                                        + "artist_id=" + artistID + "&"
                                        + "artist_name=" + escape(artistName) + "&"
                                        + "exclusive=" + exclusive + "&"
                                        + "comm_rating=" + commRating + "&"
                                        + "new=" + newStatus + "&"
                                        + "origin=" + escape(SongData.originText(origin, djName,
djPosessive)) + "&"
```

```
                                        + "popular=" + popular + "&"
                                        + "song_rating=" + songRating + "&"
                                        + "song_rating_type=1" + "&"
                                        + "album_rating=" + albumRating + "&"
                                        + "album_rating_type=1" + "&"
                                        + "artist_rating=" + artistRating + "&"
                                        + "artist_rating_type=1"
                                        + djs

                                        + fans(conn, songID)
                                        + radioStations(conn, userID, songID)
                                        + "&ticker_text=&image_url=" // not used
                                );

                        volume = request.getParameter("volume");
                        saveVolume(conn, userID, volume);
                        conn.close();
                }
                catch (DBException e)
                {
                        System.err.println("DBException: " + e.getMessage());
                        e.printStackTrace();
                }
                catch (Exception e)
                {
                        out.println("Exception raised: " + e);
                        e.printStackTrace();
                }
                out.close();
        }
        private void saveVolume(DBConnection conn, String userID, String volumeStr) throws
DBException
        {
                if (volumeStr == null)
                        return;
                double volume = 0;
                try
                {
                        Double dblVolume = new Double(volumeStr);
                        if (dblVolume != null)
                                volume = dblVolume.doubleValue();
                }
                catch (Exception e)
                {
                        return;
                }
                if (volume > 0 && volume <= 100)
                {
                        conn.executeSQL("exec sp_lcSetVolume_isux " + userID + ", " + volume);
                }
        }
        private String djRatings(DBConnection conn, String userID, int itemID, String storedProc, String
variableName) throws DBException
        {
                String result = "";
                String djName;
                String ratingStr;
                int rating;
                int count = 1;
                DBResultSet rs = conn.executeSQL("exec " + storedProc + " " + userID + ", " + itemID);
                while (!rs.getBOF() && !rs.getEOF())
                {
```

```
                                        rating = rs.getInt("rating");
                                        if (rating <= 0)
                                        {
                                                ratingStr = "X";
                                        }
                                        else
                                        {
                                                ratingStr = "" + rating;
                                        }
                                        result = result.concat(
                                                "&" + variableName + "_name"  + count + "=" +
escape(rs.getString("alias"))
                                                + "&" + variableName + "_id"    + count + "=" + rs.getInt("userID")
                                                + "&" + variableName + "_value"  + count + "=" + ratingStr
                                                + "&" + variableName + "_online" + count + "=" +
isOnline(rs.getInt("minutesSincePlay"))
                                        );
                                        count++;
                                        rs.next();
                                }

                                return result;
                        }

                        private String djRatings(DBConnection conn, String userID, int itemID, byte itemType) throws
DBException
                        {
                                if (itemType == Constants.ITEM_TYPE_SONG)
                                {
                                        return djRatings(conn, userID, itemID,
"sp_lcGetUserDJRatingsForSongID_xsxx", "dj_rating");
                                }
                                else if (itemType == Constants.ITEM_TYPE_ALBUM)
                                {
                                        return djRatings(conn, userID, itemID,
"sp_lcGetUserDJRatingsForAlbumID_xsxx", "dj_rating");
                                }
                                else if (itemType == Constants.ITEM_TYPE_ARTIST)
                                {
                                        return djRatings(conn, userID, itemID,
"sp_lcGetUserDJRatingsForArtistID_xsxx", "dj_rating");
                                }

                                return "";
                        }

                        private String radioStations(DBConnection conn, String userID, int songID) throws DBException
                        {
                                int count = 0;
                                String result = "";
                                DBResultSet rs = conn.executeSQL("exec
sp_lcGetSubscribedBDSStationsPlayingSong_xsxx " + userID + ", " + songID);
                                while (!rs.getBOF() && !rs.getEOF())
                                {
                                        result = result.concat(
                                                "&radio_id"  + count + "=" + rs.getInt("bdsStationID")
                                                + "&radio_name" + count + "=" + escape(rs.getString("callLetters") + "
" + rs.getString("description"))
                                        );
                                        count++;
                                        rs.next();
                                }
```

```
                return result;
        }
        private String fans(DBConnection conn, int songID) throws DBException
        {
                String result = "";
                int count = 1;
                int rating;
                String ratingStr = "";
                DBResultSet rs = conn.executeSQL("exec sp_lcGetFans_xsxx " + songID);
                while (!rs.getBOF() && !rs.getEOF() && count <= 5)
                {
                        result = result.concat(
                                "&fan_name"   + count + "=" + escape(rs.getString("alias"))
                                + "&fan_id"    + count + "=" + rs.getInt("userID")
                                + "&fan_online" + count + "=" +
isOnline(rs.getInt("minutesSincePlay"))
                                );
                        count++;
                        rs.next();
                }
                if (count > 1 && !rs.getEOF())
                {
                        result = result.concat("&fan_id" + count + "=0" + "&fan_name" + count +
"=more...");
                }
                return result;
        }

        private String formatAlbumYear(int year)
        {
                if (year > 0)
                {
                        return " (" + year + ")";
                }
                return "";
        }
        private int isExclusive(String albumName)
        {
                if (albumName != null)
                {
                        if (albumName.indexOf("Launch Live") > -1)
                        {
                                return 1;
                        }
                }
                return 0;
        }
        private int isOnline (int lastPlay)
        {
                if (ONLINE_TIMEOUT > lastPlay)
                {
                        return 1;
                }
                return 0;
        }
        private int isPopular (int commRating)
        {
                if (commRating > Constants.POPULAR_THRESHOLD)
                {
                        return 1;
                }
                return 0;
```

```
        }
        private int isNew (Date dateAdded)
        {
                if (dateAdded == null)
                {
                        return 0;
                }
                long twoWeeks = Util.MILLISECONDS_IN_SECOND *
                                                Util.SECONDS_IN_MINUTE    *
                                                Util.MINUTES_IN_HOUR      *
                                                Util.HOURS_IN_DAY         *
                                                14;
                Date now = new Date();
                if (now.getTime() - dateAdded.getTime() < twoWeeks)
                {
                        return 1;
                }
                return 0;
        }
        private String escape(String thing)
        {
                if (thing == null)
                {
                        return "";
                }
                return URLEncoder.encode(thing);
        }
        public void init (ServletConfig config)
                throws ServletException
        {
                super.init(config);
        }
        public void destroy()
        {
        }
}
/* eof */
```

GetSongRatingsFromDB

```
package com.launch.PlaylistGenerator;
import java.util.*;
public final class GetSongRatingsFromDB extends Thread
{
        private Vector userIDs;
        private Vector results;
    //-------------------------------------------------------------
        public GetSongRatingsFromDB(Vector userIDs, Vector results)
        {
                this.userIDs = userIDs;
                this.results = results;
        }
        public void run()
        {
                Thread.currentThread().setName("GetSongRatingsFromDB");
                Util.debug(Thread.currentThread().getName() + " thread started");
                Date startDate = new Date();

                try
                {
                        String sql = "SELECT iUserID_FK userID, iSongID_FK songID, iRating rating
FROM a200SongRating WHERE iUserID_FK IN (" + RatingsCache.GetVectorAsCommaDelimitedList(userIDs) +
')';
                        DBConnection conn = new DBConnection();
                        DBResultSet rs = conn.executeSQL(sql);
                        CachedRating cr;
                        while (!rs.getBOF() && !rs.getEOF())
                        {
                                cr = new CachedRating(rs.getInt("userID"), rs.getInt("songID"),
(byte)rs.getInt("rating"), Constants.ITEM_TYPE_SONG);
                                results.addElement(cr);
                                rs.next();
                        }
                        conn.close();
                }
                catch (DBException oops)
                {
                        System.err.println("DBException in GetSongRatingsFromDB: " +
oops.getMessage());
                }
                Util.printElapsedTime(Thread.currentThread().getName(), startDate);
        }
}
```

IntHash

```
package com.launch.PlaylistGenerator;
import java.util.Hashtable;
/**
 * A hashtable that uses ints as keys and values.
 */
public class IntHash extends Hashtable
{
	public synchronized int get(int key)
	{
		Object thing = get(new Integer(key));

		if (thing == null)
			return 0;
		else
			return ((Integer) thing).intValue();
	}
	public synchronized int put(int key, int value)
	{
		put(new Integer(key), new Integer(value));

		return value;
	}

	private synchronized int change(int key, int valueChange)
	{
		return put(key, get(key) + valueChange);
	}

	public synchronized int increment(int key)
	{
		return change(key, 1);
	}
	public synchronized int decrement(int key)
	{
		return change(key, -1);
	}
	public synchronized int increment(int key, int howMuch)
	{
		return change(key, howMuch);
	}
	public synchronized int decrement(int key, int howMuch)
	{
		return change(key, -howMuch);
	}


}
```

IntHash.java Page 1 of 1 11/05/99 1:26 PM

Item

```
package com.launch.PlaylistGenerator;
public class Item
{

        public final static byte TYPE_ANY    = 0;
        public final static byte TYPE_ALBUM  = 1;
        public final static byte TYPE_ARTIST = 2;
        public final static byte TYPE_UNKNOWN = 10;

        public int itemID;
        public Rating userRating;
        private boolean songAvgScoreCalculated = false;

        private double songAvgScore;

        // the average rating from all djs for this tiem
        public AverageRating djsAverage;

        // average rating of all songs by an artist
        public AverageRating songAverage;

        public double songAverageScore(ArtistInfo info)
        {

                if (!songAvgScoreCalculated)
                {
                        songAvgScoreCalculated = true;

                        double songsByArtist = Math.min(info.songs.size(),
Constants.MAX_SONGS_BY_ARTIST);
                        double songsRated    = Math.min(songAverage.count(),
Constants.MAX_SONGS_BY_ARTIST);

                        // deviation from the average
                        songAvgScore = ((songAverage.get() - Constants.DEFAULT_RATING)
                                * (songsRated / songsByArtist)) + Constants.DEFAULT_RATING;
                }

                return songAvgScore;
        }

        public boolean inGenres = false;

        public byte getType()
        {
                if (itemID == 0)
                        return TYPE_UNKNOWN;
                else if (itemID < 1000000)
                        return TYPE_ALBUM;
                else
                        return TYPE_ARTIST;
        }

        public String typeName()
        {
                byte type = getType();

                if (type == TYPE_ALBUM)
                        return "Album";
```

```
                else if (type == TYPE_ARTIST)
                        return "Artist";
                else
                        return "Unknown";

        }

        public Item()
        {
                userRating  = new Rating();
                djsAverage  = new AverageRating();
                songAverage = new AverageRating();
        }

        public Item(int itemID)
        {
                this();
                this.itemID = itemID;
        }

        public String toString(SongInfoCache cache)
        {

                String title = "(Not available)";
                byte type = getType();


                if (type == TYPE_ARTIST)
                {
                        ArtistInfo artist = (ArtistInfo) cache.get(itemID, SongInfoCache.TYPE_ARTIST);

                        if (artist != null)
                                title = artist.title;
                }
                else if (type == TYPE_ALBUM)
                {
                        AlbumInfo album = (AlbumInfo) cache.get(itemID, SongInfoCache.TYPE_ALBUM);

                        if (album != null)
                                title = album.title;
                }

                return typeName() + " \"" + title + "\" (" + itemID + ") "
                        + "user=" + userRating.toString()
                        + " djs=" + djsAverage.toString()
                  + " songAverage=" + songAverage.toString()
                        + " songAvgScore=" + songAvgScore;

        }

}
```

ItemsProfile

```
package com.launch.PlaylistGenerator;
import java.util.Hashtable;
import java.util.Enumeration;
import javax.servlet.ServletOutputStream;
public class ItemsProfile
{

        private Hashtable hash;

        public ItemsProfile()
        {
                hash = new Hashtable();
        }

        public synchronized Item get(int itemID)
        {
                return get(new Integer(itemID));
        }

        public synchronized Item get(Integer itemID)
        {
                return (Item) hash.get(itemID);
        }

        /**
         * puts a new item in the hash and returns it.
         * If it's already there, just return it
         */
        public synchronized Item put(int itemID)
        {
                Integer ID = new Integer(itemID);

                Item it = get(ID);

                if (it == null)
                {
                        it = new Item(itemID);
                        hash.put(ID, it);
                        return it;
                }
                else
                        return it;
        }
        public void print(ServletOutputStream out, SongInfoCache cache)
        {
                for (Enumeration e = hash.keys(); e.hasMoreElements() ;) {

                        Item anItem = get((Integer) e.nextElement());

                        Util.out(out, anItem.toString(cache));
                }

        }

        public String inList(byte type)
        {
                String list = "";

                for (Enumeration e = hash.keys(); e.hasMoreElements() ;) {
```

```
                        Item anItem = get((Integer) e.nextElement());

                        if (type == Item.TYPE_ANY || anItem.getType() == type)
                        {
                                list = list.concat(anItem.itemID + ",");
                        }
                }
                // remove that extra comma
                if (list.length() > 0)
                        list = list.substring(0, list.length() - 1);

                return list;

        }
}
```

Media

```
package com.launch.PlaylistGenerator;
public class Media
{
        int mediaID;
        short mediaType;
        String filepath;

        public Media(int mediaID, short mediaType, String filepath)
        {
                this.mediaID   = mediaID;
                this.mediaType = mediaType;
                this.filepath  = filepath;
        }

        public String toString()
        {
                return mediaType + ": " + mediaID;
        }
        public static short getMediaType(Bandwidth speed, MediaFormat format)
        {
                if (format.get() == MediaFormat.WINDOWSMEDIA)
                {
                        if (speed.get()     == Bandwidth.SPEED_28)
                                return 211;
                        else if (speed.get() == Bandwidth.SPEED_56)
                                return 147;
                        else if (speed.get() >= Bandwidth.SPEED_100)
                                return 212;
                        else
                                return 0;
                }

                return 0;
        }

        public static Bandwidth typeToBandwidth(short mediaType)
        {
                if (mediaType == 211)
                        return new Bandwidth(Bandwidth.SPEED_28);
                else if (mediaType == 147)
                        return new Bandwidth(Bandwidth.SPEED_56);
                else if (mediaType == 212)
                        return new Bandwidth(Bandwidth.SPEED_100);

                return new Bandwidth();
        }

}
```

Media.java Page 1 of 1 11/05/99 1:28 PM

MediaFormat

```
package com.launch.PlaylistGenerator;
public class MediaFormat
{
        public final static byte WINDOWSMEDIA = 1;
        public final static byte REALMEDIA    = 2;
        public final static byte QUICKTIME    = 3;

        private boolean beenset = false;

        private byte value;

        // when we start supporting more than one format, just take this out
        public MediaFormat()
        {
                value = WINDOWSMEDIA;
                beenset = true;
        }

        public MediaFormat(byte format)
        {
                value = format;
                beenset = true;
        }

        public byte get()
        {
                return value;
        }

        public void set(byte format)
        { .
                value = format;
                beenset = true;
        }

        public boolean isSet()
        {
                return beenset;
        }

        public String toString()
        {
                if (value == WINDOWSMEDIA)
                        return "WindowsMedia";
                else if (value == REALMEDIA)
                        return "RealMedia";
                else if (value == QUICKTIME)
                        return "QuickTime";

                return "UNKNOWN";
        }
}
```

MediaGatewayServlet

```
package com.launch.PlaylistGenerator;
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
/**
*-------------------------------------------------------------
* PlaylistGeneratorServlet.java 8/16/99
* Servlet that redirects to media
* Copyright (c) 1999 Launch, Inc.
* @author Jeff Boulter
*-------------------------------------------------------------
*/
public final class MediaGatewayServlet extends HttpServlet
{
        /** what browser signature we look for */
        private static final String mpSignature = "NSPlayer";
        /** when we get an unauthorized browser, play this */
        private static final String unauthorizedBrowser = "audio/errors/unauthorizedbrowser.asf";
        /** when we get an unauthorized user, play this */
        private static final String unauthorizedUser = "audio/errors/unauthorizeduser.asf";
        /** when we get an unauthorized user, play this */
        private static final String outOfMedia = "audio/errors/outofmedia.asf";
        /** how many tries we take to get media */
        private static final int MAX_ITERATIONS = 5;
        /** this is the header that media player uses toe indicate which query it is */
        private static final String CONTEXT_TAG = "request-context=";
        /** To work around a problem with reading multiple headers with the same name in servlet 2.0 + jrun, we
look for these headers to determine the context */
        private static final String FIRST_REQUEST_PRAGMA  = "xClientGUID";
        private static final String SECOND_REQUEST_PRAGMA = "stream-switch-entry";
        private static final String REQUEST_CONTEXT = "request-context=";
        private static final int STREAMING_MEDIA_TIMEOUT=1000*60*15;
        /**
         * Handle requests...
         */
        public final void doGet (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
        {
//              Util.debug("MediaRedirectServlet:doGet() received a request");
                DBConnection conn = null;
                ServletOutputStream out = null;
                int context;
                int userID = -1;
                boolean debug=false;
                try
                {
                        // get connections and streams
                        conn = new DBConnection();
                        out  = response.getOutputStream();
                        // get parameters from http
                        debug = (request.getParameter("ralph") != null);
                        // setup response data
                        setResponseHeaders(response);
                        setResponseContentType(response, debug);
                        // get parameters from http
                        userID = Integer.parseInt(request.getParameter("u"));
                        if (!checkUserAgent(request.getHeader("USER_AGENT"), debug, out))
```

```
                {
                        return;
                }
                // muck with clip and clip schedule
                ClipSchedule schedule = new ClipSchedule(userID);
                schedule.init(conn);  //db call 1
                Clip aClip = null;
                int iteration;
                boolean done = false;
                // keep going until we get a good path

                for (iteration = 0; iteration < MAX_ITERATIONS && !done; iteration++)
                {
                        aClip = new Clip(schedule.nextClipType(debug, out));

                        if (aClip == null || aClip.type() == Clip.TYPE_NONE)
                        {
                                done = true;
                                System.err.println("user " + userID + " is out of songs to play");

                        }
                        else
                        {
                                // get the paths and stuff
                                aClip.getPath(conn, schedule); // db call 2
                                if (aClip.isSet())
                                {
                                        done = true;
                                }
                                else
                                {
                                        done = true;
                                        System.err.println("user " + userID + " is out of media of type
    " + aClip.typeName() + " to play");
                                }
                        }
                }

                // update the playlist
                schedule.playlist.save(conn, userID); // db call 3

                if (aClip == null)
                        out.println(Constants.STREAM_SERVER + "/" + outOfMedia);
                else
                {

                        // log the play
                        aClip.logPlay(conn, userID); // db call 4

                        // get the URL
                        out.println(aClip.URL());
                }
        }
        catch (NumberFormatException e)
        {
                out.println("Bad userId");
                // print out the MMS path to redirect to
                if (debug)
                {
                        out.println("redirecting to " + unauthorizedUser);
                }
                else
```

```
                {
                        out.println(Constants.STREAM_SERVER + "/" + unauthorizedUser);
                }
        }
        catch (Throwable e)
        {
                System.err.println("Generic Exception in MediaGateway for userID " + userID + ": " +
e.getMessage());
                e.printStackTrace();
        }
        finally
        {
                try
                {
                        if (out!=null)
                        {
                                out.close();
                        }
                        if (conn!=null)
                        {
                                conn.close();
                        }
                }
                catch (SocketException se)
                {
                        // don't do anything, the person disconnected, no error, (or mediaplayer sampled
first 32 bytes.)
                }
                catch (Exception e1)
                {
                        e1.printStackTrace();
                }
        }
}
private final boolean checkUserAgent(String agent, boolean debug, ServletOutputStream out) throws
IOException
 {
        if (!(agent!=null && agent.startsWith(mpSignature)))
        {
                if (debug)
                {
                        out.println("invalid useragent. Would stream " + unauthorizedBrowser);
                        return true;
                }
                else
                {
                        out.println(Constants.STREAM_SERVER + "/" + unauthorizedBrowser);
                }
                return(false);
        }
        else
        {
                return(true);
        }
}
private final void setResponseContentType(HttpServletResponse response, boolean debug)
{
        if (debug)
        {
                response.setContentType("text/plain");
        }
        else
```

```
185                     {
                                response.setContentType("video/x-ms-asf");
                        }
                }
                private final void setResponseHeaders(HttpServletResponse response)
190             {
                        response.setHeader("Pragma", "no-cache");
                        response.setHeader("Cache-control", "no-cache");
                        response.setHeader("Expires", "0");
                }
195     /*
                private static final void readFileToOutputStream(String filename, HttpServletResponse response, boolean
        debug)
                {
                        readFileToOutputStream(new File(filename), response, debug);
200             }
                private static final void readFileToOutputStream(File the_file, HttpServletResponse response, boolean
        debug)
                {
                        try
205                     {
                                BufferedInputStream bis=new BufferedInputStream(new FileInputStream(the_file));
                                BufferedOutputStream bos=new BufferedOutputStream(response.getOutputStream());
                                bos.flush(); //this is to ward off any problems I think there might be a jrun problem with
        initializing the output stream fast enough, i.e. before we get there...
210                             BufferedWriter br=new BufferedWriter(new OutputStreamWriter(bos));
                                if (debug)
                                        Util.out(response.getOutputStream(), "streaming file " + the_file + " of size " +
        the_file.length());
                                else
215                                     response.setContentLength((int)the_file.length());
                                // System.err.println("streaming file " + the_file + " of size " + the_file.length());
                                RedirectStream redirecting_stream=new RedirectStream(bis, bos, debug,
        response.getOutputStream());
                                redirecting_stream.start();
220                             redirecting_stream.join(STREAMING_MEDIA_TIMEOUT, 0);
                                if (redirecting_stream.isAlive()) redirecting_stream.stop();
                                //System.err.println("finished streaming");
                        }
                        catch (SocketException se)
225                     {
                                // don't do anything, the person disconnected, no error, (or mediaplayer sampled first 32
        bytes.)
                        }
                        catch (FileNotFoundException fe)
                        {
230                             System.err.println("readFileToOutputStream could not find file " + the_file + " for
        reading:" + fe.getMessage());
                        }
                        catch (Exception e)
235                     {
                                e.printStackTrace();
                        }
                }
                private int getContext(HttpServletRequest request)
240             {
                        try
                        {
                                String pragma = request.getHeader("pragma");
        //                      Util.debug("pragma is " + pragma);
                                if (pragma == null)
245                                     return 0;
```

```
                        int index = pragma.indexOf(REQUEST_CONTEXT);
//                      Util.debug("index is " + index);
                        if (index < 0)
                        {
                                return 0;
                        }
                        else
                        {
                                int start = index + REQUEST_CONTEXT.length();
                                String contextNum = pragma.substring(start, start + 1);
//                              Util.debug("contextNum is " + contextNum);
                                return Integer.parseInt(contextNum);
                        }
// when I can read multiple headers with the same name I should use the below code
//                      int location=pragma.indexOf(CONTEXT_TAG);
//                      location=location+CONTEXT_TAG.length();
//                      int last_location;
//                      for (last_location=location; last_location<pragma.length() &&
pragma.charAt(last_location)!=','; last_location++);
//                      return(Integer.parseInt(pragma.substring(location, last_location)));
                }
                catch (Exception e)
                {
                        Util.debug("Exception caught in getContext: " + e.toString());
                        return 0;
                }
        }
*/
}
```

MediaGatewayServlet.java Page 7 of 7 11/05/99 1:24 PM

MediaList

```
package com.launch.PlaylistGenerator;
import java.util.Vector;
public class MediaList
{
	private Vector media = new Vector(0, 1);

	public void add(short mediaType, int mediaID, String filepath)
	{
		media.addElement(new Media(mediaID, mediaType, filepath));
	}

	public boolean inType(short mediaType)
	{

		Media test;

		for (int i = 0; i < media.size(); i++)
		{

			test = (Media) media.elementAt(i);

			if (test.mediaType == mediaType)
				return true;
		}

		return false;
	}

	public int getID(short mediaType)
	{
		for (int i = 0; i < media.size(); i++)
		{
			Media aMedia = (Media) media.elementAt(i);

			if (aMedia.mediaType == mediaType)
				return aMedia.mediaID;
		}

		return 0;
	}

	public String getFilepath(short mediaType)
	{
		for (int i = 0; i < media.size(); i++)
		{
			Media aMedia = (Media) media.elementAt(i);

			if (aMedia.mediaType == mediaType)
				return aMedia.filepath;
		}

		return null;
	}

	public int size()
	{
		return media.size();
	}

```

```
        public Media typeAt(int index)
        {
                return (Media) media.elementAt(index);
        }

        public String toString()
        {
                String result = "";

                if (media == null)
                        return "(none)";

                for (int i = 0; i < media.size(); i++)
                {
                        result = result.concat(media.elementAt(i).toString() + ",");
                }
                return "(" + result + ")";
        }

}
```

MediaList.java Page 2 of 2 11/05/99 1:28 PM

PickCount

```
package com.launch.PlaylistGenerator;
import javax.servlet.ServletOutputStream;
/**
 */
public class PickCount
{
	int explicit;
	int implicit;
	int unrated;
	String method = "";
	public PickCount(int userID, int djID, int ratio, int playlistSize, Population songs, ServletOutputStream
out)
	{
		float explicitSize = songs.explicit.size();
		float implicitSize = songs.implicit.size();
		float unratedSize  = songs.unrated.size();
		Util.out(out, "Available: explicit songs: " + explicitSize + ", implicit songs: " + implicitSize + ",
unrated songs: " + unratedSize);
		Util.out(out, "Ratio: " + ratio);
		// if you're listening to someone else's station, try to not listen to any unrated songs
		if (userID == djID)
		{
			// let's try to use their ratio
			double totalRated = (explicitSize + implicitSize);
			if (totalRated < Constants.MIN_RATINGS_TO_HONOR_RATIO)
			{
				method = "New User Unrated Ratio";
				ratio = Constants.NEW_USER_UNRATED_RATIO;
			}
			int maxPlicit = (int) Math.round(playlistSize * (100 - ratio) * 0.01);
			int maxRatedToPick = (int) Math.round(explicitSize *
Constants.MAX_PERCENT_RATED_SONGS_TO_PICK * 0.01);
			// pick three times as much from rated
			int explicitToPick = (int) Math.round(playlistSize * (100 - ratio) * 0.01 * (explicitSize /
totalRated) * 3);
			int implicitToPick = maxPlicit - explicitToPick;
			explicit   = (int) Math.min(maxRatedToPick, explicitToPick);
			implicit   = (int) Math.min(implicitSize, implicitToPick);
			// pick up the slack in unrated
			unrated = (playlistSize - explicit - implicit);
			method = "Unrated Ratio";
		}
		// if you're listening to someone else's station and they have enough ratings,
		// don't play unrated
		else if ((explicitSize + implicitSize) > Constants.MIN_SIZE_FOR_NO_UNRATED)
		{
			explicit = (int) Math.round(playlistSize * 0.50);
			explicit = (int) Math.round(Math.min(explicit, (explicitSize *
Constants.MAX_PERCENT_RATED_SONGS_TO_PICK) * 0.01));
			implicit = (int) Math.min(playlistSize, implicitSize) - explicit;
			method = "DJ play - no unrated";
			// if we didn't get enough, use the default method
			if (explicit + implicit < playlistSize)
			{
				explicit = (int) Math.round(playlistSize * 0.33);
				explicit = (int) Math.round(Math.min(explicit, (explicitSize *
Constants.MAX_PERCENT_RATED_SONGS_TO_PICK) / 100.0));
				implicit = (int) Math.round(playlistSize * 0.33);
				implicit = (int) Math.round(Math.min(implicit, (implicitSize *
```

```
Constants.MAX_PERCENT_RATED_SONGS_TO_PICK) / 100.0));
                                unrated = playlistSize - explicit - implicit;
                                method = "DJ play - not enough rated";
                        }
                }
                // if neither of these worked
                else
                {
                        explicit = (int) Math.round(playlistSize * 0.33);
                        explicit = (int) Math.round(Math.min(explicit, (explicitSize *
Constants.MAX_PERCENT_RATED_SONGS_TO_PICK) / 100.0));
                        implicit = (int) Math.round(playlistSize * 0.33);
                        implicit = (int) Math.round(Math.min(implicit, (implicitSize *
Constants.MAX_PERCENT_RATED_SONGS_TO_PICK) / 100.0));
                        unrated = playlistSize - explicit - implicit;
                        method = "Default 33/33/33 method";
                }
                Util.out(out, "Picking: explicit songs: "
                                        + explicit
                                        + ", implicit songs: "
                                        + implicit
                                        + ", unrated songs: "
                                        + unrated
                                        + ", method = " + method
                                );
        }
        public String toString()
        {
                return "explicit to pick: "
                        + explicit
                        + ", implicit to pick: "
                        + implicit
                        + ", unrated to pick: "
                        + unrated;
        }
        public void reset()
        {
                explicit = 0;
                implicit = 0;
                unrated  = 0;
        }
}
```

PickCount.java Page 3 of 3 11/05/99 1:24 PM

PickList

```
package com.launch.PlaylistGenerator;
import java.util.Vector;
public class PickList extends Vector
{
        public PickList(PickCount counts)
        {
                // make a list of all the song types that we need to pick
                for (int i = 0; i < counts.explicit; i++)
                        addElement(Song.EXPLICIT);
                for (int i = 0; i < counts.implicit; i++)
                        addElement(Song.IMPLICIT);
                for (int i = 0; i < counts.unrated; i++)
                        addElement(Song.UNRATED);
        }
        public void addElement(short value)
        {
                addElement(new Short(value));
        }
        public void reAdd (short type, Vector songGroup, Population songs)
        {
                // try to pick from the same bucket again
                if (songGroup.size() > 0)
                        addElement(type);
                // otherwise, try the other ones
                else if (songs.explicit.size() > 0)
                        addElement(Song.EXPLICIT);
                else if (songs.implicit.size() > 0)
                        addElement(Song.IMPLICIT);
                else if (songs.unrated.size() > 0)
                        addElement(Song.UNRATED);
        }
        public short getRandom()
        {
                if (size() < 0)
                        return 0;
                int lucky = (int) Util.random(size() - 1);
                // figure out what group to pick from
                short type = ((Short) elementAt(lucky)).shortValue();
                removeElementAt(lucky);
                return type;
        }
}
```

PickList.java Page 2 of 2 11/05/99 1:27 PM

PickStatus

```
package com.launch.PlaylistGenerator;
public class PickStatus
{
    public final static int NOT_PICKED = 0;
    public final static int REJECTED = 2;
    public final static int PICKED = 1;

    int status;
    int order = -1;
    short percentile;

    public String toString()
    {
        return toDisplayString(Util.DISPLAY_TEXT);
    }

    public String toDisplayString(int displayType)
    {
        String redStart = "";
        String greenStart = "";
        String fontEnd = "";

        if (displayType == Util.DISPLAY_HTML)
        {

            redStart = "<FONT COLOR=red><B>";
            greenStart = "<FONT COLOR=green><B>";
            fontEnd  = "</B></FONT>";
        }

        switch (status) {
            case NOT_PICKED:
                return "N  ";
            case PICKED:
                return greenStart + " P " + fontEnd;
            case REJECTED:
                return redStart + "  R" + fontEnd;
            default:
                return "   ";
        }
    }
}
```

PickStatus.java Page 1 of 1 11/05/99 1:26 PM

PlayDataHash

```
package com.launch.PlaylistGenerator;
import java.util.Enumeration;
public class PlayDataHash extends IntHash
{
        public String toString()
        {
                String myString = "";


                for (Enumeration e = keys(); e.hasMoreElements() ;) {
                        // debug.write("interation " + i++);
                        int stationID = ((Integer) e.nextElement()).intValue();
                        int rank = get(stationID);
                        myString = myString.concat(
                                                "stationID: " +
                                                stationID +
                                                "=" +
                                                rank +
                                                "\n");
                }

                return myString;
        }
}
```

PlayDataHash.java Page 1 of 1 11/05/99 1:26 PM

PlayDates

```
package com.launch.PlaylistGenerator;
import java.util.Hashtable;
import java.util.Date;
import java.util.Enumeration;
import java.text.SimpleDateFormat;
import java.io.InputStreamReader;
import java.text.ParsePosition;
import java.io.IOException;
import java.util.Calendar;
public class PlayDates
{

        private static final String dateFormat = "yyyy-MM-dd HH:mm:ss";

        private Hashtable hash;
        int userID;

        double secondsInDay = Util.MILLISECONDS_IN_SECOND *
                                                Util.SECONDS_IN_MINUTE *
                                                Util.MINUTES_IN_HOUR *
                                                Util.HOURS_IN_DAY;
        // for date parsing
        private static StringBuffer year   = new StringBuffer("1234");
        private static StringBuffer month  = new StringBuffer("12");
        private static StringBuffer day    = new StringBuffer("12");

        private static StringBuffer hour   = new StringBuffer("12");
        private static StringBuffer minutes = new StringBuffer("12");
        public Date dbDate = new Date();

        private boolean loaded = false;

        public PlayDates()
        {
                hash = new Hashtable();
        }

        public void put(int songID, Date lastPlayed)
        {

                // the common case is that they will have NOT played this song before,
                // so create the Integer object in anticipation that we will use it for
                // the put as well.

                Integer i = new Integer(songID);

                Date before = get(i);

                // save only the most recent play of a song

                if (before == null || before.getTime() < lastPlayed.getTime())
                {
                        hash.put(i, lastPlayed);
                }
        }

        public Date get(int songID)
        {
                return (Date) hash.get(new Integer(songID));
```

```
        }

        public Date get(Integer songID)
        {
                return (Date) hash.get(songID);
        }

        public Enumeration keys()
        {
                return hash.keys();
        }

        public void remove(Integer songID)
        {
                hash.remove(songID);
        }

        public int size()
        {
                return hash.size();
        }

        public String toString()
        {

                String result = "";

                for (Enumeration e = hash.keys(); e.hasMoreElements() ;) {
                        Integer songID = (Integer) e.nextElement();
                        Date playedAt = get(songID);
                        result = result.concat("{" + songID + " = " + playedAt + "} ");
                }

                return result;
        }

        public String toDBString()
        {
                Date startDate = new Date();

                StringBuffer buffer = new StringBuffer(100000);

                Calendar cal = Calendar.getInstance();

                Integer songID;
                Date playedAt;

                for (Enumeration e = hash.keys(); e.hasMoreElements() ;) {
                        songID = (Integer) e.nextElement();
                        playedAt = get(songID);

//                      System.out.println(playedAt);

                        cal.setTime(playedAt);

                        buffer.append(cal.get(Calendar.YEAR) + "-"
                                        + leadingZero(cal.get(Calendar.MONTH) + 1) + "-"
                                        + leadingZero(cal.get(Calendar.DAY_OF_MONTH)) + " "
                                        + leadingZero(cal.get(Calendar.HOUR_OF_DAY)) + ":"
                                        + leadingZero(cal.get(Calendar.MINUTE)) + ":00=" +
songID + ",");
```

```
//                  result = result.concat(formatter.format(playedAt) + "=" + songID + ",");
                }

                Util.printElapsedTime("toDBString", startDate);

                return buffer.toString();
        }

        public static final String leadingZero (int value)
        {
                if (value < 10)
                        return "0" + value;

                return value + "";
        }

        public float getScore(Integer songID)
        {
                Date lastPlayed = get(songID);

                if (lastPlayed == null)
                        return 0;

                double secondsSincePlayed = new Date().getTime() - lastPlayed.getTime();
                double daysSincePlayed = secondsSincePlayed / secondsInDay;
                double logValue = Math.log(daysSincePlayed + 0.01);
                return (float) Math.min(100, (22.0 * logValue));
        }
        public void save(DBConnection conn)
        {
//              Date dateStarted = new Date();

                if (!loaded)
                        return;

                try
                {
                        conn.executeUpdate("exec sp_lcSavePlayHistoryText_isux " + userID + ", '" +
toDBString() + "'", false);
                }
                catch (DBException e)
                {
                        System.err.println("DBException in PlayDates:save: " + e.toString());
                }

//              Util.printElapsedTime("save", dateStarted);
        }

        public void markRecentlyPlayed(SongInfoCache cache, Population songs)
        {

                double now = dbDate.getTime();
                double lastThreeHours = Util.MILLISECONDS_IN_SECOND *
                                                        Util.SECONDS_IN_MINUTE *
                                                        Util.MINUTES_IN_HOUR *
                                                        3;
                Integer songID;
                Date playedAt;
                SongInfo info;
                int artistID, albumID;
                for (Enumeration e = hash.keys(); e.hasMoreElements() ;)
                {
```

```
                songID = (Integer) e.nextElement();
                playedAt = get(songID);

                if (now - playedAt.getTime() < lastThreeHours)
                {
                        // mark songs played in the last three hours
                        // so as to comply with the RIAA rules
                        // and make sure we don't pick too many later
                        info = (SongInfo) cache.get(songID, SongInfoCache.TYPE_SONG);

                        if (info != null)
                        {
                                artistID = info.getArtistID();
                                albumID  = info.getAlbumID();

                                // "various artists" albums don't count
                                if (!ArtistInfo.isVariousArtists(artistID))
                                {
                                        songs.artistCounts.increment(artistID);
                                }
                                songs.albumCounts.increment(albumID);
                        }
                }
        }
}

public void oldLoad(DBConnection conn, int userID)
{
        this.userID = userID;

        try
        {
                String sql = "exec sp_lcoGetLastPlayed_xsxx " + userID;
                DBResultSet rs   = conn.executeSQL(sql);

                loaded = true;

                Date lastDate;
                int songID;
                while (!rs.getBOF() && !rs.getEOF())
                {
                        songID   = rs.getInt("songID");
                        lastDate = rs.getTimestamp("lastPlayed");

                        put(songID, lastDate);

                        rs.next();
                }

        }
        catch (DBException e)
        {
                System.err.println("DBException in PlayDates.oldLoad: " + e.toString());
        }

}

public void load(DBConnection conn, int userID)
{
```

```
                Date startDate = new Date();

                // be careful of the SQL Server TEXTSIZE parameter which is by default 64KB

                this.userID = userID;

                double aDay = Util.MILLISECONDS_IN_SECOND *
                                        Util.SECONDS_IN_MINUTE *
                                        Util.MINUTES_IN_HOUR *
                                        Util.HOURS_IN_DAY;

                double aMonth = aDay * Util.DAYS_IN_MONTH;

                try
                {

                        String sql = "exec sp_lcGetSongHistoryText_xsxx " + userID;

                        DBResultSet rs    = conn.executeSQL(sql);
                        Util.printElapsedTime("LP: ran getsonghistorytext", startDate);

                        if (!rs.getBOF() && !rs.getEOF())
                        {
                                loaded = true;
                                char[] stuff = new char[100000];

                                InputStreamReader reader = new
InputStreamReader(rs.getAsciiStream("played"));

                                Util.printElapsedTime("LP: created reader", startDate);
                                dbDate = rs.getTimestamp("dbDate");
                                long dbDateTime = dbDate.getTime();

                                reader.read(stuff);
                                Util.printElapsedTime("LP: read into stuff", startDate);
                                Calendar cal = Calendar.getInstance();
                                int lastStart = 0;
                                int songID = 0;
//                              SimpleDateFormat formatter1 = new
SimpleDateFormat(PlayDates.dateFormat);
//                              ParsePosition pos = new ParsePosition(0);

                                Date datePlayed = null;
                                String parseme = new String();

                                long length = stuff.length;

                                for (int i = 0; i < length; i++)
                                {

                                        switch (stuff[i])
                                        {
                                        case '=':
//              //                              parseme = new String(stuff, lastStart, i - lastStart);
//              //                              pos.setIndex(0);
//              //                              datePlayed = formatter1.parse(parseme, pos);

                                                datePlayed = parseDate(stuff, lastStart, cal);
//      //                                      System.out.println("date is " + datePlayed);
//      //                                      if (datePlayed == null)
//      //                                      {
//      //                                              pos.setIndex(0);
```

```
//                                                      datePlayed = formatter2.parse(parseme, pos);
//                                                  }
                                                    lastStart = i + 1;
                                                    break;

                                            case ',':
                                                    parseme = new String(stuff, lastStart, i - lastStart);

                                                    try
                                                    {
                                                            songID = Integer.parseInt(parseme);
                                                    }
                                                    catch (NumberFormatException e) { }

                                                    // save 'em
                                                    // also don't save them if they're > 30 days old
                                                    if (songID > 0 && datePlayed != null && ((dbDateTime -
datePlayed.getTime()) < aMonth))

                                                    {
                                                            put(songID, datePlayed);
                                                    }
                                                    songID = 0; // reset
                                                    datePlayed = null; // reset

                                                    lastStart = i + 1;
                                                    break;

                                            case 0:
                                                    // we're at the end of the string
                    //                              Util.printElapsedTime("LP: found null at char " + i,
startDate);
                                                    return;
                                            }
                                    }
                            }
                    }
                    catch (DBException oops)
                    {
                            Util.debug("DBException in PlayDates.load: " + oops.getMessage());
                    }
                    catch (IOException oops)
                    {
                            Util.debug("IOException in PlayDates.load: " + oops.getMessage());
                    }
            }
            /**
             * Why? Because SimpleDateFormat is *way* too slow.
             */
            private final Date parseDate(char[] chars, int start, Calendar cal)
            {
                    // 1999-10-13 17:19:00
                    // 0123456789012345678
                    /*
                    String year, month, day, hour, minutes;
                    year   = new String(chars, start, 4);
                    month  = new String(chars, start + 5, 2);
                    day    = new String(chars, start + 8, 2);

                    hour   = new String(chars, start + 11, 2);
                    minutes = new String(chars, start + 14, 2);
                    */
```

```
                year.setCharAt(0, chars[start + 0]);
                year.setCharAt(1, chars[start + 1]);
                year.setCharAt(2, chars[start + 2]);
                year.setCharAt(3, chars[start + 3]);

                month.setCharAt(0, chars[start + 5]);
                month.setCharAt(1, chars[start + 6]);

                day.setCharAt(0, chars[start + 8]);
                day.setCharAt(1, chars[start + 9]);

                hour.setCharAt(0, chars[start + 11]);
                hour.setCharAt(1, chars[start + 12]);

                minutes.setCharAt(0, chars[start + 14]);
                minutes.setCharAt(1, chars[start + 15]);

                int yearInt = 0, monthInt = 0, dayInt = 0, hourInt = 0, minutesInt = 0;
        //      try
        //      {
                        yearInt   = parseInt(year);
                        monthInt  = parseInt(month);
                        dayInt    = parseInt(day);

                        hourInt   = parseInt(hour);
                        minutesInt = parseInt(minutes);
        //      }
        //      catch (NumberFormatException e) { return null;}

        //      cal.clear();
                cal.set(yearInt, monthInt - 1, dayInt, hourInt, minutesInt, 0);
                return cal.getTime();
        }

        private static final int parseInt(StringBuffer s)
        {
                int result = 0;
                int last = s.length() - 1;

                for (int i = last; i >= 0; i--)
                {
                        result += char2int(s.charAt(i)) * Math.pow(10, last - i);
                }

                return result;
        }

        private final static int char2int(char ch)
        {
                switch (ch)
                {
                        case '1':
                                return 1;
                        case '2':
                                return 2;
                        case '3':
                                return 3;
                        case '4':
                                return 4;
                        case '5':
                                return 5;
```

```
                                case '6':
                                        return 6;
                                case '7':
                                        return 7;
                                case '8':
                                        return 8;
                                case '9':
                                        return 9;
                                default:
                                        return 0;
                        }
                }
        }
```

Playlist

```
package com.launch.PlaylistGenerator;
import java.util.Vector;
import java.util.Hashtable;
import java.util.Enumeration;
import java.util.Date;
public class Playlist
{
	Vector media;
	Vector news;
	Vector ads;
	Vector tips;
	int ID;
	int userID;
	int djID;
	int moodID;
	short mediaType;
	boolean debug;
	boolean popularOnly = false;
	PickCount counts;
	public final static int BUCKET_COUNT = 5;
	private int lastIndex;
	int buckets[];
	IntHash artists;
	IntHash albums;
	public Playlist()
	{
		artists = new IntHash();
		albums  = new IntHash();
		counts  = null;
		media   = new Vector();
		news    = new Vector();
		ads     = new Vector();
		tips    = new Vector();
		buckets = new int[BUCKET_COUNT];
		lastIndex = -1;
		debug = false;
	}
	public Playlist(int playlistID)
	{
		this();
		ID = playlistID;
	}
	public void resetSources()
	{
		for (int i = 0; i < BUCKET_COUNT; i++)
			buckets[i] = 0;
	}
	private void saveOrigins(DBConnection conn)
	{
		String listString = "";
		SongData data;
		for (int i = 0; i < media.size(); i++)
		{
			listString = listString.concat(((SongData) media.elementAt(i)).originTclList());
		}
		try
		{
			conn.executeSQL("exec sp_lcSaveOrigins_ixxd " + userID + ", '" + listString + "'");
		}
```

```
                catch (DBException oops)
                {
                        Util.debug("DB Exception: " + oops.getMessage());
                }
        }
        public Playlist2 toPlaylist2()
        {
                Playlist2 result = new Playlist2();
                // copy playlist
                for (int i = 0; i < media.size(); i++)
                {
                        result.songs.addElement(((SongData) media.elementAt(i)).toPlaylistEntry(mediaType));
                }
                // copy news
                for (int i = 0; i < news.size(); i++)
                {
                        result.news.addElement(((Clip) news.elementAt(i)).toPlaylistEntry(mediaType));
                }
                // copy ads
                for (int i = 0; i < ads.size(); i++)
                {
                        result.ads.addElement(((Clip) ads.elementAt(i)).toPlaylistEntry(mediaType));
                }
                // copy tips
                for (int i = 0; i < tips.size(); i++)
                {
                        result.tips.addElement(((Clip) tips.elementAt(i)).toPlaylistEntry(mediaType));
                }
                return result;
        }
        public String toString()
        {
                IntHash artistCount      = new IntHash();
                IntHash albumCount       = new IntHash();
                IntHash querySource      = new IntHash();
                Hashtable querySourceName = new Hashtable();
                IntHash originSource     = new IntHash();
                Hashtable originSourceName = new Hashtable();
                Hashtable artistNames    = new Hashtable();
                Hashtable albumNames     = new Hashtable();
                String result = "Playlist " + ID + " for userID " + userID
                                                + " (djID " + djID + ") in mood " + moodID
                                                + " with mediaType " + mediaType
                                                + ", pickCounts: " + counts
                                                + " has " + media.size() + " songs:"
                                                + Util.newLine;
                for (int i = 0; i < media.size(); i++)
                {
                        SongData data = (SongData) media.elementAt(i);
                        String songStr = data.getMediaID(mediaType) + " "
                                + data.getAlbumID() + " "
                                + data.getArtistID() + " "
                                + data.songID + " "
                                + data.getArtistName() + " "
                                + data.getAlbumName() + " "
                                + data.getSongName() + Util.newLine;
                        querySource.increment(data.querySource);
                        querySourceName.put(new Integer(data.querySource),
data.sourceString(data.querySource));
                        byte origin = data.origin();
                        originSource.increment(origin);
                        originSourceName.put(new Integer(origin), data.sourceString(origin));
```

```
                artistCount.increment(data.getArtistID());
                albumCount.increment(data.getAlbumID());
                if (data.getArtistName() != null)
                        artistNames.put(new Integer(data.getArtistID()), data.getArtistName());
                if (data.getAlbumName() != null)
                        albumNames.put(new Integer(data.getAlbumID()), data.getAlbumName());
                result = result.concat(songStr);
        }
        result = result.concat(Util.newLine);
        for (Enumeration e = artistCount.keys(); e.hasMoreElements() ;) {
                int artistID = ((Integer) e.nextElement()).intValue();
                String artistStr = artistCount.get(artistID)
                                                        + " songs are by the artist "
                                                        + artistNames.get(new Integer(artistID))
                                                        + " (" + artistID + ") "
                                                + Util.newLine;
                result = result.concat(artistStr);
        }
        result = result.concat(Util.newLine);
        for (Enumeration e = albumCount.keys(); e.hasMoreElements() ;) {
                int albumID = ((Integer) e.nextElement()).intValue();
                String albumStr = albumCount.get(albumID)
                                                        + " songs are from the album "
                                                        + albumNames.get(new Integer(albumID))
                                                        + " (" + albumID + ") "
                                                + Util.newLine;
                result = result.concat(albumStr);
        }
        result = result.concat(Util.newLine);
        for (Enumeration e = querySource.keys(); e.hasMoreElements() ;) {
                int source = ((Integer) e.nextElement()).intValue();
                int songCount = querySource.get(source);
                double doubleCount = new Double(songCount).doubleValue();
                String str = songCount
                                                        + " songs ("
                                                        + ((doubleCount / length()) * 100)
                                                        + "%) are from the "
                                                        + querySourceName.get(new
Integer(source))
                                                        + " query"
                                                + Util.newLine;
                result = result.concat(str);
        }
        result = result.concat(Util.newLine);
        for (Enumeration e = originSource.keys(); e.hasMoreElements() ;) {
                int source = ((Integer) e.nextElement()).intValue();
                int songCount = originSource.get(source);
                double doubleCount = new Double(songCount).doubleValue();
                String str = songCount
                                                        + " songs ("
                                                        + ((doubleCount / length()) * 100)
                                                        + "%) originated from "
                                                        + originSourceName.get(new
Integer(source))
                                                + Util.newLine;
                result = result.concat(str);
        }
        result = result.concat(Util.newLine);
        int bucketSize = 100 / BUCKET_COUNT;
        double playlistLength = media.size();
        for (int i = 0; i < BUCKET_COUNT; i++)
        {
```

```
                        result = result.concat(
                                                "Percentile "
                                                + (i * bucketSize) + "% - "
                                                + ((i + 1) * bucketSize) + "%: " + buckets[i] + " ("
                                                + Util.fix(100 * (buckets[i] / playlistLength), 2, 0) + "%)" +
Util.newLine);
                }
                return (result + Util.newLine);
        }
        public int length ()
        {
                return media.size();
        }
        public void append (SongData song)
        {
                float bucketSize = (new Float(101)).floatValue() / (new Float(BUCKET_COUNT)).floatValue();
                int bucket = (int) Math.floor(song.status.percentile / bucketSize);
//              Util.debug("adding mediaID " + song.mediaID
//                              + " in percentile " + song.status.percentile + " (bucket "
//                              + bucket + ")");
                media.addElement(song);
                buckets[bucket]++;
        }
        public Playlist shuffle()
        {
                Vector newList = new Vector(media.size());
                int rand = 0;
                while (media.size() > 0)
                {
                        rand = (int) Util.random(media.size() - 1);
                        Object m = media.elementAt(rand);
                        media.removeElementAt(rand);
                        newList.addElement(m);
                }
                media = newList;
                return this;
        }
        public int nextOrdinal(DBConnection conn)
        {
                int ordinal = 1;
                try
                {
                        DBResultSet rs = conn.executeSQL("exec sp_IcGetOrdinalID_xsxx " + userID);
                        while (!rs.getBOF() && !rs.getEOF())
                        {
                                ordinal = rs.getInt("ordinal");
                                rs.next();
                        }
                        conn.executeSQL("exec sp_IcUpdatePlaylistData_ixxd "
                                                                + userID + ", "
                                                                + djID + ", "
                                                                + moodID + ", "
                                                                + mediaType);
                }
                catch (DBException oops)
                {
                        Util.debug("DB Exception in Playlist::nextOrdinal: " + oops.getMessage());
                }
                return ordinal;
        }
        public void deleteHighOrdinals(DBConnection conn, int ordinal)
        {
```

```
            try
            {
                conn.executeSQL("exec sp_lcDeletePlaylistRange_xxxd "
                                + userID + ", "
                                + ordinal);
            }
            catch (DBException oops)
            {
                Util.debug("DB Exception in Playlist::deleteHighOrdinals: " + oops.getMessage());
            }
        }
        private SimplePlaylist toSimplePlaylist()
        {
            SimplePlaylist result = new SimplePlaylist();
            result.mediaType = this.mediaType;
            result.djID     = this.djID;
            result.moodID   = this.moodID;

            // copy playlist
            for (int i = 0; i < media.size(); i++)
            {
                result.songs.addElement(((SongData) media.elementAt(i)).toSimpleClip(mediaType));
            }
            // copy news
            for (int i = 0; i < news.size(); i++)
            {
                result.news.addElement(((Clip) news.elementAt(i)).toSimpleClip(mediaType));
            }
            // copy ads
            for (int i = 0; i < ads.size(); i++)
            {
                result.ads.addElement(((Clip) ads.elementAt(i)).toSimpleClip(mediaType));
            }
            // copy tips
            for (int i = 0; i < tips.size(); i++)
            {
                result.tips.addElement(((Clip) tips.elementAt(i)).toSimpleClip(mediaType));
            }
            return result;
        }

        public void save (DBConnection conn, SimplePlaylist oldPlaylist)
        {
            Date startDate = new Date();

            SimplePlaylist thoreau = toSimplePlaylist();

            Util.printElapsedTime("Convert to SimplePlaylist", startDate);

            if (oldPlaylist != null)
            {
                thoreau.lastAd   = oldPlaylist.lastAd;
                thoreau.lastNews = oldPlaylist.lastNews;
                thoreau.lastTip  = oldPlaylist.lastTip;
            }

            thoreau.save(conn, userID);

            Util.printElapsedTime("SavePlaylist", startDate);
        }

        /*
```

```
        public boolean save (DBConnection conn)
        {
                if (length() <= 0)
                        return false;
                boolean resetOrdinal = false;
                int highOrdinal, ordinal;
                Date startDate = new Date();
                highOrdinal = ordinal = nextOrdinal(conn);
                if (highOrdinal > MAX_ORDINAL)
                {
                        ordinal = 1;
                        resetOrdinal = true;
                }
                Util.printElapsedTime("GetOrdinal", startDate);
                Thread saveNews = new SaveClips(news, "sp_lcSaveNewsPlaylist_ixxd", ordinal, mediaType,
userID);
                Thread saveAds  = new SaveClips(ads, "sp_lcSaveAdsPlaylist_ixxd", ordinal, mediaType,
userID);
                Thread saveTips = new SaveClips(tips, "sp_lcSaveTipsPlaylist_ixxd", ordinal, mediaType,
userID);
                int partition = (int) Math.round(media.size() / 4.0);
                Thread savePlaylist1 = new SavePlaylist(this, 0, partition, ordinal);
                Thread savePlaylist2 = new SavePlaylist(this, partition, partition * 2, ordinal + partition);
                Thread savePlaylist3 = new SavePlaylist(this, partition * 2, partition * 3, ordinal + (partition * 2));
                Thread savePlaylist4 = new SavePlaylist(this, partition * 3, media.size(), ordinal + (partition * 3));
                savePlaylist1.start();
                savePlaylist2.start();
                savePlaylist3.start();
                savePlaylist4.start();
                saveNews.start();
                saveAds.start();
                saveTips.start();
                deleteHighOrdinals(conn, highOrdinal - 1);
                // everybody done yet?
                saveOrigins(conn);
                try
                {
                        saveNews.join();
                        saveAds.join();
                        saveTips.join();
                        savePlaylist1.join();
                        savePlaylist2.join();
                        savePlaylist3.join();
                        savePlaylist4.join();
                }
                catch (InterruptedException e)
                {
                        Util.debug("Playlist::save was interrupted while waiting");
                }
                Util.printElapsedTime("SavePlaylist", startDate);
                return true;
        }
        */

        private void saveClips(DBConnection conn, Vector clips, String storedProc)
        {
                for (int i = 0; i < clips.size(); i++)
                {
                        Clip aClip = (Clip) clips.elementAt(i);
                        String sql = "exec " + storedProc + " "
                                + ID + ", "
                                + aClip.mediaID + ", "
```

```
                                + mediaType + ", "
                                + userID;
                        try
                        {
                                DBResultSet rs = conn.executeSQL(sql);
                        }
                        catch (DBException oops)
                        {
                                Util.debug("DB Exception: " + oops.getMessage());
                        }
                }
        }
        public String newLine()
        {
                return Util.newLine;
        }
        public String toASX()
        {
                String asx = "<ASX VERSION=\"3.0\" PREVIEWMODE=\"NO\">" + Util.newLine
                        + Util.tab() + "<REPEAT>" + Util.newLine;
                String streamURL = Constants.STREAM_URL + "?u="
                        + userID;
                for (int i = 0; i < 10; i++)
                {
                        asx = asx.concat(Util.tab(2) +
                                                "<ENTRY>" + Util.newLine
                                                + Util.tab(3)
                                                + "<REF HREF=\""
                                                + streamURL
                                                + "&n="
                                                + i
                                                + ".asp"
                                                + "\"/>" + Util.newLine
                                                + Util.tab(2)
                                                + "</ENTRY>" + Util.newLine);
                }
                asx = asx.concat(Util.tab() + "</REPEAT>" +Util.newLine
                                                + "</ASX>" + Util.newLine);
                return asx;
        }
}
```

Playlist.java Page 10 of 10 11/05/99 1:38 PM

Playlist2

```
package com.launch.PlaylistGenerator;
import java.util.*;
//---------------------------------------------------------------
/**
* @author Ted Leung
* @version 1999-09-22
**/
//---------------------------------------------------------------
public final class Playlist2 implements java.io.Serializable
{
        //*************************************************************************
        // variables
        //*************************************************************************
                /** all these vectors contain exclusively Strings which are directory/filename of audio files */
                public Vector songs;
                public Vector news;
                public Vector ads;
                public Vector tips;

        //*************************************************************************
        // methods
        //*************************************************************************

                public Playlist2()
                {
                        songs = new Vector(50);
                        news  = new Vector(10);
                        ads   = new Vector(10);
                        tips  = new Vector(10);
                }

        //---------------------------------------------------------------
        /**
        **/
        //---------------------------------------------------------------
                public final String toString()
                {
                        return
                        (
                                "songs="+songs.toString() + ", " +
                                "news="+news.toString() + ", " +
                                "ads="+ads.toString() + ", " +
                                "tips="+tips.toString()
                        );
                }
        //*************************************************************************
}
```

PlaylistCreatorTest

```
package com.launch.PlaylistGenerator;
public class PlaylistCreatorTest
{
        public static void main(String[] args)
        {

                Util.debug("using database server " + Constants.DB_SERVER);

                SongInfoCache songCache = new SongInfoCache(null);
                songCache.ratingsCache = new RatingsCache();
//              PlaylistParameters params = new PlaylistParameters(3771, null, 0, 13302);
                PlaylistParameters params = new PlaylistParameters(6474126, null, 0, 6474126);
                PlaylistGenerator gen    = new PlaylistGenerator(params, songCache, null);
                Playlist playlist        = gen.create(true, null);

                gen.toMatrix(null, Util.DISPLAY_TEXT);
                System.exit(0);
        }
}
```

PlaylistCreatorTest.java Page 1 of 1 11/05/99 1:35 PM

PlaylistEntry

```
package com.launch.PlaylistGenerator;
import java.io.*;
public class PlaylistEntry implements Serializable
{

        public String title, filepath, songTitle, albumTitle, artistTitle;
        public int mediaID, songID, albumID, artistID;

        public short implicit;
        public byte origin;


}
```

PlaylistEntry.javaPage 1 of 1 11/05/99 1:28 PM

PlaylistGenerator

```
package com.launch.PlaylistGenerator;
import java.util.Vector;
import java.util.Date;
import javax.servlet.ServletOutputStream;
import java.util.Enumeration;
public class PlaylistGenerator
{
        public final static byte RATER_DJ   = 1;
        public final static byte RATER_BDS  = 2;
        public final static byte RATER_GENRE = 3;
        private short factor     = (short)Constants.DEFAULT_PICK_FACTOR;
        private short ratio      = (short) Constants.DEFAULT_UNRATED_RATIO;
        private int playlistSize  = Constants.DEFAULT_PLAYLIST_SIZE;
        private int playlistID;
        private boolean haveTitles = false;
        private Date startDate;
        private Date lastDate;
        private int userID;
        private int djID;
        private int moodID;
        private short mediaType;
        private IntHash ratings;
        private ItemsProfile items;
        private PlayDates lastPlayed;
        private Population songs;
        private Vector news;
        private Vector ads;
        private Vector tips;
        private DJList djs;
        private GenreList genres;
        private Bandwidth speed;
        private MediaFormat format;

        private StationList stations;
        private ServletOutputStream out;
        private SongInfoCache songCache;
        private boolean playExplicitLyrics = true;
        /**
         * Creates a new playlist generator.
         */
        public PlaylistGenerator()
        {
                songs     = new Population();
                news      = new Vector();
                ads       = new Vector();
                tips      = new Vector();
                ratings   = new IntHash();
                djs       = new DJList();
                items     = new ItemsProfile();
                lastPlayed = new PlayDates();
                genres    = new GenreList();
                stations  = new StationList();
        }
        public PlaylistGenerator (PlaylistParameters params, SongInfoCache cache, ServletOutputStream out)
        {
                this();
                userID     = params.userID;
                moodID     = params.moodID;
                djID       = params.djID;
```

```
        if (djID <= 0) djID = userID;

        speed      = params.speed;
        format     = params.format;
        playlistSize = params.playlistSize;
        songCache  = cache;
        this.out   = out;
    }
    private void getRandom()
    {
        Date startDate = new Date();
        Song ditty;
        SongData data;
        SongInfo info;
        SongList songList;
        int rowCount = 0;
        double pickCount;
        double totalSongs;
        // the simple way
        /*
        songList = cache.getInGenres(genres);
        pickCount = Math.min(songList.size(), this.RANDOM_SONGS_COUNT);
        // import them all
        if (pickCount == songList.size())
        {
            for (int i = 0; i < pickCount; i++)
            {
                info = songList.elementAt(i);
                rowCount += addRandom(info, SongData.SOURCE_RANDOM);
            }
        }
        // import a random subset
        else
        {
            for (int i = 0; i < pickCount; i++)
            {
                info = songList.pickRandom();
                rowCount += addRandom(info, SongData.SOURCE_RANDOM);
            }
        }
        */
        // the faster(?) but way more complicated way
        int songCount  = songCache.countInGenres(genres);
        totalSongs     = songCache.size(SongInfoCache.TYPE_SONG);
        double percent = (songCount / totalSongs) * 100.0;
        Util.printElapsedTime("GetRandom done counting in genres", startDate);

        // the problem is if we pick randomly and they want songs from
        // only a few genres, we're probably not going to get enough to create
        // a playlist. So instead, if there's not a whole lot of songs in those genres,
        // just get them directly from the genres instead of taking our chances with random
        Util.debug("getRandom: " + songCount + " non-unique songs in genres (" + percent + "%)");
        if (percent < Constants.MIN_SONGS_IN_GENRES_TO_GET_RANDOM)
        {
            Util.debug("getRandom: getting directly from genres");
            // get the list of songs from each genre
            // choose the number to pick from each, proportional to the number of songs
            // pick them
            int totalToPick = Math.min(Constants.RANDOM_SONGS_COUNT, songCount);
            for (int i = 0; i < genres.size(); i++)
            {
```

```
                                songList = songCache.getInGenre(genres.genreAt(i));
                                pickCount = totalToPick * (songList.size() / ((double) songCount));
                                for (int j = 0; j < pickCount; j++)
                                {
                                        info = songList.pickRandom();
                                        if (info != null)
                                        {
                                                rowCount += addRandom(info,
SongData.SOURCE_GENRES);
                                        }
                                }
                        }
                }
                else
                {
                        Util.debug("getRandom: picking randomly from all songs");
                        for (int i = 0; i < Constants.RANDOM_SONGS_COUNT; i++)
                        {
                                // this is really fast
                                info = songCache.randomSong();
                                // this is really slow
                                rowCount += addRandom(info, SongData.SOURCE_RANDOM);
                        }
                }
                Util.debug("getRandom added " + rowCount + " songs");
                Util.printElapsedTime("GetRandom done", startDate);
        }
        private int addRandom(SongInfo info, byte source)
        {
                SongData data = songs.initSongGetData(info.songID, Song.UNRATED);
                if (data != null)
                {
                        data.querySource = source;
                        data.setInfo(info);
                        return 1;
                }
                return 0;
        }
        private void getPopular(SongList list)
        {
                Date startDate = new Date();
                Song ditty;
                SongData data;
                SongInfo info;
                int rowCount = 0;
                if (list != null)
                {
                        for (int i = 0; i < list.size(); i++)
                        {
                                info = list.elementAt(i);
                                data = songs.getSongData(info.songID);
                                if (data != null)
                                {
                                        // we can't add it, but let's append the info while we're here
                                        data.setInfo(info);
                                }
                                else
                                {
                                        data = songs.initSongGetData(info.songID, Song.UNRATED);
                                        if (data != null)
                                        {
                                                data.querySource = data.SOURCE_POPULAR;
```

```
                                        data.setInfo(info);
                                    }
                                    rowCount++;
                                }
                            }
                        }
                        Util.debug("getPopular added " + rowCount + " songs");
                }
                /**
                 * Gets all the required media and data to generate a playlist.
                 */
                private void gatherMedia(DBConnection conn)
                {
                        Thread getLastPlayed  = new GetLastPlayed(lastPlayed, userID, out);
                        Util.out(out, "starting gathering threads at " + timeStamp());
                        // try to start them in ascending order of speed
                        getLastPlayed.start();
                        // get djs, genres, and bds subscriptions
                        getSubscriptions(conn, djID, moodID);
                        Util.out(out, "getSubscriptions done " + timeStamp());
                        // we need to wait for the djs to come in first
                        Thread getRatings = new GetRatings(songs, items, djID, djs, songCache, out);
                        getRatings.start();
                        Util.out(out, "All threads started " + timeStamp());
                        // getpopular and getrandom should not be threads since they are purely processor bound now
                        getPopular(songCache.getPopular(mediaType));
                        Util.out(out, "getPopular done " + timeStamp());
                        getRandom();
                        Util.out(out, "getRandom done (picked " + Constants.RANDOM_SONGS_COUNT + " songs)" +
timeStamp());
                        Util.out(out, "genres for mood " + moodID + ":" + genres.toString());
                        // wait for them to finish
                        try
                        {
                                getRatings.join();
                                getLastPlayed.join();
                        }
                        catch (InterruptedException oops)
                        {
                                Util.debug("InterruptedException: " + oops.toString());
                        }
                        Util.out(out, "gatherMedia done " + timeStamp());
                }
                public void getSubscriptions(DBConnection conn, int userID, int moodID)
                {
                        Date started = new Date();
                        try
                        {
                                DBResultSet rs = conn.executeSQL("exec sp_IcoGetAllSubscriptions_xsxx "
                                                                                + userID + ", "
                                                                                + moodID);
                                int raterID;
                                int raterType;
                                while (!rs.getBOF() && !rs.getEOF())
                                {
                                        raterID   = rs.getInt("raterID");
                                        raterType = rs.getInt("raterType");
                                        if (raterType == RATER_DJ)
                                        {
                                                djs.addElement(new DJ(raterID));
                                        }
                                        else if (raterType == RATER_GENRE)
```

```
                    {
                        genres.add((short) raterID);
                    }
                    else if (raterType == RATER_BDS)
                    {
                        stations.addElement(new Station(raterID));
                    }

                    rs.next();
                }
                Util.debug("getSubscriptions added "
                        + djs.size() + " DJs, "
                        + genres.size() + " Genres, "
                        + stations.size() + " Stations");
            }
            catch (DBException oops)
            {
                Util.debug("DB Exception in getSubscriptions " + oops.getMessage());
            }
            Util.printElapsedTime("getSubscriptions took ", started);
        }
        /**
        Calculates scores for all the songs and puts them into the various vectors
        */
        public void processSongs()
        {
            byte result;
            WeightMatrix weights = new WeightMatrix();
            Integer songID;
            Song aSong;
            SongData data;
            short type;
            Date playedAt;
            SongInfo info;
            int good = 0;
            int tested = 0;
            int artistID, albumID;
            Item albumItem;
            Item artistItem;

            AlbumArtistData albumAndArtist = new AlbumArtistData();

            IntHash reasons = new IntHash();
            double now = lastPlayed.dbDate.getTime();
            double lastThreeHours = Util.MILLISECONDS_IN_SECOND *
                                    Util.SECONDS_IN_MINUTE *
                                    Util.MINUTES_IN_HOUR *
                                    3;
            for (Enumeration e = songs.keys(); e.hasMoreElements() ;)
            {
                tested++;

                albumAndArtist.reset();

                songID = (Integer) e.nextElement();
                aSong  = songs.get(songID);
                data   = aSong.getData();

                if (aSong.getType() == Song.EXCLUDED)
                {
                    reasons.increment(1);
                }
```

```
                else
310             {
                    // add the song info
                    info = data.getInfo();
                    // get the song info from the cache
                    if (info == null)
315                 {
                        info = (SongInfo) songCache.get(songID,
    SongInfoCache.TYPE_SONG);
                        data.setInfo(info);
                    }
320                 // if it's still null, it's not encoded
                    if (info == null)
                    {
                        aSong.setType(Song.EXCLUDED);
                        reasons.increment(2);
325                     continue;
                    }
                    // ok, we have the song info.
                    // add last played
                    playedAt = lastPlayed.get(songID);
330                 if (playedAt != null)
                    {
                        lastPlayed.remove(songID);

                        // don't play the same song twice in a 3 hour period
335                     if (now - playedAt.getTime() < lastThreeHours)
                        {
                            // mark songs played in the last three hours
                            // so as to comply with the RIAA rules
                            // and make sure we don't pick too many later
340                         artistID = data.getArtistID();
                            albumID  = data.getAlbumID();
                            // "various artists" albums don't count
                            if (!ArtistInfo.isVariousArtists(artistID))
                            {
345                             songs.artistCounts.increment(artistID);
                            }
                            songs.albumCounts.increment(albumID);
                            // make sure we don't play this again so soon
                            aSong.setType(Song.EXCLUDED);
350                         reasons.increment(3);
                            continue;
                        }
                        data.lastPlayed = lastPlayed.getScore(songID);
                    }
355                 // check for bad words
                    if (!playExplicitLyrics && info.hasExplicitLyrics())
                    {
                        aSong.setType(Song.EXCLUDED);
                        reasons.increment(4);
360                     continue;
                    }
                    // now check for media in the type we need
                    if (!info.media.inType(mediaType))
                    {
365                     aSong.setType(Song.EXCLUDED);
                        reasons.increment(5);
                        continue;
                    }
                    // check for valid genres
370                 if (!info.album.inGenres(genres))
```

```
                {
                        // for popular songs, don't exclude them,
                        // otherwise we won't be able to default to them
                        // if the genre restrictions are too tight
                        if (data.querySource == data.SOURCE_POPULAR)
                        {
                                songs.remove(songID);
                        }
                        reasons.increment(6);
                        aSong.setType(Song.EXCLUDED);
                        continue;
                }
                // we got this far, so try to calculate an implicit rating
                result = data.calculateImplicit(items, albumAndArtist);
                if (result == SongData.EXCLUDE_ME)
                {
                        aSong.setType(Song.EXCLUDED);
                        reasons.increment(7);
                        continue;
                }
                if (result == SongData.MAKE_ME_IMPLICIT)
                {
                        aSong.setType(Song.IMPLICIT);
                        data.calculateDJs(items, albumAndArtist);
                        data.score(weights, stations);
                        songs.implicit.addElement(data);
                        good++;
                }
                else
                {
                        type = aSong.getType();
                        // put the song in a list to pick from later
                        if (type == Song.EXPLICIT)
                        {
                                // your djs don't matter if you explicitly rated the song
                                songs.explicit.addElement(data);
                        }
                        else if (type == Song.IMPLICIT)
                        {
                                data.calculateDJs(items, albumAndArtist);
                                songs.implicit.addElement(data);
                        }
                        else if (type == Song.UNRATED)
                        {
                                data.calculateDJs(items, albumAndArtist);
                                songs.unrated.addElement(data);
                        }
                        // calculate the score
                        data.score(weights, stations);
                        good++;
                }
        }
}

Util.out(out, "scores calculated " + timeStamp());

// for all the songs we didn't get for whatever reason, make sure we
// are accounting for their plays for compliance with RIAA rules
lastPlayed.markRecentlyPlayed(songCache, songs);
Util.out(out, "recently played albums and artists marked " + timeStamp());

Util.out(out, "Of " + tested + " songs, these are the reasons for exclusion: "
```

```
                                + reasons.get(1) + " were already excluded, "
                                + reasons.get(2) + " were not encoded, "
                                + reasons.get(3) + " were played in the last 3 hours, "
                                + reasons.get(4) + " had explicit lyrics, "
                                + reasons.get(5) + " were not in mediaType " + mediaType + ", "
                                + reasons.get(6) + " were not in their genres, "
                                + reasons.get(7) + " had an implicit rating of 0.");
            Util.out(out, "There are " + good + " songs available for play");
        }
        /**
         * Gets a user's preferences for their playlists
         */
        public boolean getOptions(DBConnection conn)
        {
            int rowCount = 0;
            short tempRatio;
            short bandwidth = 0;
            // returns: ratio, factor, mediaType
            String sql = "exec sp_IcGetPreferences_xsxx " + userID;
            try
            {
                DBResultSet rs = conn.executeSQL(sql);
                if (!rs.getBOF() && !rs.getEOF())
                {
                    tempRatio = (short) rs.getInt("unratedQuota");
                    if (tempRatio > 0 && tempRatio < 100)
                        ratio = tempRatio;
                    playExplicitLyrics = rs.getBoolean("explicit");
                    // if there was no mediatype set from the parameters
                    // set it to the default
                    if (!speed.isSet())
                        speed.set(rs.getShort("bandwidth"));

                    rowCount++;
                }
            }
            catch (DBException oops)
            {
                Util.debug("DB Exception in getOptions: " + oops.getMessage());
            }
            mediaType = Media.getMediaType(speed, format);
            Util.debug("Play dirty songs?: " + playExplicitLyrics);
            Util.debug("Bandwidth: " + speed.toString());
            Util.debug("Format: " + format.toString());
            Util.debug("mediaType: " + mediaType);
            return (rowCount > 0);
        }
        /**
         * Creates a playlist.
         */
        public Playlist createPlaylist(DBConnection conn)
        {
            Util.out(out, "start of createPlaylist " + timeStamp());
            Playlist playlist = new Playlist(playlistID);
            gatherMedia(conn);
            processSongs();
            playlist = makePlaylist(factor, ratio, playlistSize, playlist);
            Util.out(out, "end of createPlaylist " + timeStamp());
            return playlist;
        }
        private void logCreate(DBConnection conn)
        {
```

```
                try
                {
                        conn.executeSQL("exec sp_lcLogPlaylist_ixxx "
                                                + userID + ", "
                                                + djID + ", "
                                                + moodID + ", "
                                                + 0 + ", "
                                                + mediaType + ", "
                                                + elapsedTime()
                                                );
                }
                catch (DBException e)
                {
                        Util.debug("DBException in logCreate: " + e.toString());
                }
        }
        /**
         * Creates and immediately saves a playlist.
         */
        public Playlist create(boolean save, SimplePlaylist oldPlaylist)
        {
                DBConnection conn = null;
                Playlist playlist = null;
                try
                {
                        conn = new DBConnection();
                        getOptions(conn);
                        playlist = createPlaylist(conn);
                        Util.out(out, "starting to save playlist " + timeStamp());
                        if (save)
                                playlist.save(conn, oldPlaylist);
                        logCreate(conn);
                        Util.out(out, "done saving playlist " + timeStamp());
                        conn.close();
                }
                catch (DBException oops)
                {
                        Util.out(out, "DBException in create: " + oops.getMessage());
                }
                catch (Throwable e)
                {
                        System.err.println("Generic Exception caught in PlaylistGenerator: " + e.toString());
                        e.printStackTrace();
                }
                return playlist;
        }
        public Playlist makePlaylist(int factor, int ratio, int playlistSize, Playlist playlist)
        {
                Util.out(out, "ordering..." + timeStamp());
                songs.sort(songs.explicit);
                songs.sort(songs.implicit);
                songs.sort(songs.unrated);
                Util.out(out, "finished sorting vectors at " + timeStamp());
                playlist.counts = new PickCount(userID, djID, ratio, playlistSize, songs, out);
                // set up the playlist
                playlist.userID    = this.userID;
                playlist.moodID    = this.moodID;
                playlist.djID      = this.djID;
                playlist.mediaType = this.mediaType;
                // copy the list of albums and artists recently played
                // for the RIAA rules
                playlist.albums  = (IntHash) songs.albumCounts.clone();
```

```
                playlist.artists = (IntHash) songs.artistCounts.clone();
                // pick songs
                pickSongs(playlist);
                // check if we got everything we need
                if (playlist.media.size() < playlistSize)
                {
                        Util.out(out, "We only got " + playlist.media.size() + " songs for user " + playlist.userID
+ ". Playing popular music in mediaType " + mediaType);
                        // uh oh, we didn't get enough songs; play popular stuff
                        playlist.counts.explicit = 0;
                        playlist.counts.implicit = 0;
                        playlist.counts.unrated  = playlistSize;
                        playlist.albums  = (IntHash) songs.albumCounts.clone();
                        playlist.artists = (IntHash) songs.artistCounts.clone();

                        playlist.resetSources();
                        playlist.media.removeAllElements();
                        playlist.popularOnly = true;
                        songs.importPopular(songCache.getPopular(mediaType), lastPlayed, playExplicitLyrics);
                        pickSongs(playlist);
                }
                // pick news
                pickNews(playlist);
                Util.out(out, "picked news " + timeStamp());
                // pick ads
                pickAds(playlist);
                Util.out(out, "picked ads " + timeStamp());
                // pick tips
                pickTips(playlist);
                Util.out(out, "picked tips " + timeStamp());
                Util.out(out, "playlist has " + playlist.length() + " songs");
                Util.out(out, "shuffling playlist...");
                return playlist.shuffle();
        }
        public void pickNews(Playlist list)
        {
                list.news = songCache.randomClipList(SongInfoCache.TYPE_NEWS, mediaType,
Constants.MAX_NEWS_ITEMS);
        }
        public void pickAds(Playlist list)
        {
                list.ads = songCache.randomClipList(SongInfoCache.TYPE_AD, mediaType,
Constants.MAX_ADS);
        }
        public void pickTips(Playlist list)
        {
                list.tips = songCache.randomClipList(SongInfoCache.TYPE_TIP, mediaType,
Constants.MAX_TIPS_ITEMS);
        }
        public Playlist pickSongs (Playlist list)
        {
                Util.out(out, "start of pickSongs " + timeStamp());
                PickList pickTypes = new PickList(list.counts);
                int pickOrder = 0;
                int iteration = 0;
                int artistID, albumID, artistCount, albumCount;
                short type;
                SongData pick;
                SongGroup songGroup;
                while (pickTypes.size() > 0)
                {
                        iteration++;
```

```
                    pick     = null;
                    songGroup = null;
                    // get a group to pick from
                    type = pickTypes.getRandom();
                    if (type == Song.EXPLICIT && songs.explicit.size() > 0)
                    {
                            songGroup = songs.explicit;
                    }
                    else if (type == Song.IMPLICIT && songs.implicit.size() > 0)
                    {
                            songGroup = songs.implicit;
                    }
                    else
                    {
                            type = Song.UNRATED;
                            songGroup = songs.unrated;
                    }
                    // pick a random song from a group
                    pick = songGroup.pickRandom(factor);
                    // if we have none of that type, try another
                    if (pick == null)
                    {
                            pickTypes.reAdd(type, songGroup, songs);
                            continue;
                    }
                    artistID = pick.getArtistID();
                    albumID  = pick.getAlbumID();
                    artistCount = 0;
                    albumCount  = 0;
                    // check for RIAA compliance
                    // various artists and soundtracks don't count
                    if (!ArtistInfo.isVariousArtists(artistID))
                            artistCount = list.artists.get(artistID);
                    albumCount  = list.albums.get(albumID);
                    if (artistCount >= Constants.RIAA_MAX_SONGS_BY_ARTIST
                            || albumCount >= Constants.RIAA_MAX_SONGS_FROM_ALBUM)
                    {
                            pick.status.status = PickStatus.REJECTED;
                            // Util.debug("Song rejected by RIAA");
                            // we have too many from this artist or album. Try again.
                            pickTypes.reAdd(type, songGroup, songs);
                            continue;
                    }
                    // increment the album and artist counts
                    if (!ArtistInfo.isVariousArtists(artistID))
                            list.artists.increment(artistID);
                    list.albums.increment(albumID);
                    // add it to the playlist
                    list.append(pick);
                    pick.status.status = PickStatus.PICKED;
                    pick.status.order  = ++pickOrder;
            }
            songs.ordered = false;
            Util.out(out, "end of pickSongs " + timeStamp());
            return list;
    }
    public void toMatrix(ServletOutputStream out, int displayType)
    {
            songs.order();
            String h1begin = "";
            String h1end = "";
            if (displayType == Util.DISPLAY_HTML)
```

```
                {
                        h1begin = "<P><H1>";
                        h1end = "</H1>";
                }
                Util.out(out, h1begin + "Item Ratings" + h1end + Util.newLine);
                items.print(out, songCache);
                Util.out(out, h1begin + "Explicitly Rated Songs" + h1end + Util.newLine);
                songs.toMatrix(out, Song.EXPLICIT, displayType);
                Util.out(out, h1begin + "Implicitly Rated Songs" + h1end + Util.newLine);
                songs.toMatrix(out, Song.IMPLICIT, displayType);
                Util.out(out, h1begin + "Unrated Songs" + h1end + Util.newLine);
                songs.toMatrix(out, Song.UNRATED, displayType);
//                      + h1begin + "Excluded Songs" + h1end + Util.newLine
//                      + songs.excludedList();
        }
        public String timeStamp()
        {
                Date now = new Date();
                if (startDate == null)
                {
                        startDate = lastDate = now;
                }
                double diff = (now.getTime() - lastDate.getTime()) / 1000.0;
                double total = (now.getTime() - startDate.getTime()) / 1000.0;
                lastDate = now;
                return Util.newLine
                        + "----------" + Util.newLine
                        + diff + " lap time, " + total + " total" + Util.newLine
                        + "----------" + Util.newLine;
        }
        public double elapsedTime()
        {
                Date now = new Date();
                if (startDate == null)
                {
                        startDate = lastDate = now;
                }
                return (now.getTime() - startDate.getTime()) / 1000.0;
        }
}
```

PlaylistGeneratorServlet

```
package com.launch.PlaylistGenerator;

import java.io.*;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import java.util.*;
/**
*-------------------------------------------------------------
*
* PlaylistGeneratorServlet.java 6/30/99
* Servlet that creates LAUNCHcast playlists
* Copyright (c) 1999 Launch, Inc.
* @author Jeff Boulter
*-------------------------------------------------------------
*/
public class PlaylistGeneratorServlet extends HttpServlet {

        SongInfoCache songCache;
        Thread cacheUpdater;

        public void generatePlaylist(HttpServletRequest        request,
                HttpServletResponse        response) throws IOException
        {

                // get stream for output
                ServletOutputStream out = response.getOutputStream();
                GeneratorParameters prop = new GeneratorParameters(request);
                if (prop.debug())
                        response.setContentType("text/plain");
                else
                        response.setContentType("video/x-ms-asf");

                PlaylistParameters params = new PlaylistParameters(prop);
                PlaylistStatus status    = new PlaylistStatus(prop.userID());
                status.init(out);
                if (prop.debug())
                        out.print(status.toString());

                boolean generate = true;
                // no need to regenerate right now, use an old playlist
                if (prop.forceRefresh())
                {
                        if (prop.debug()) out.println("generating because forceRefresh is on");
                }
                else if (status.isStale())
                {
                        if (prop.debug()) out.println("generating because the playlist is stale");
                }
                else if (prop.speed().isSet() && (prop.speed().get() != status.speed.get()))
        {
                        if (prop.debug()) out.println("generating because the mediaTypes are different");
                }
                else if (prop.format().isSet() && (prop.format().get() != status.format.get()))
        {
                        if (prop.debug()) out.println("generating because the media formats are different");
```

```
                }
                else if (prop.moodID() != status.moodID)
        {
                        if (prop.debug()) out.println("generating because the moods are different");
                }
                else if (prop.djID() != status.djID)
                {
                        if (prop.debug()) out.println("generating because the djs are different");
                }
                else
                        generate = false;

                if (!generate) // we can use an old playlist
                {

                        // reset the ad, news, and tip dates

                        if (status.playlist != null)
                        {
                                status.resetDates();
                        }

                        Playlist playlist = new Playlist();
                        playlist.userID  = status.userID;

                        out.print(playlist.toASX());



                }
                else // we have to generate the playlist
                {

                        ServletOutputStream outStream = null;

                        if (prop.debug())
                        {
                                outStream = out;
                                out.println("regenerating playlist with parameters: " + params.toString() +
"<PRE>");
                                out.flush();
                        }
                        PlaylistGenerator gen = new PlaylistGenerator(params, songCache, outStream);
                        Playlist playlist = gen.create(!prop.dontsave(), null);

                        if (prop.debug())
                        {
                                out.println("</PRE>");
                                if (prop.debugFormat() == Util.DISPLAY_TEXT)
                                        out.println("<PRE>");
                                out.println(playlist.toString()
                                                                + "<P>");
                                if (prop.matrix())
                                {
//                                      out.println("<FONT SIZE=-1>");
                                        gen.toMatrix(out, prop.debugFormat());
//                                      out.println("</FONT>");
                                }
                                if (prop.debugFormat() == Util.DISPLAY_TEXT)
                                        out.println("</PRE>");
                                out.println("<XMP>" + playlist.toASX() + "</XMP>");
                        }
```

```
                else
                        out.print(playlist.toASX());

        }

        out.close();
    }
    public void refreshPlaylist(HttpServletRequest      request,
        HttpServletResponse     response) throws IOException
    {
        // get stream for output
        ServletOutputStream out = response.getOutputStream();
        response.setContentType("text/plain");
        // this is the stuff coming in on the query string
        GeneratorParameters prop = new GeneratorParameters(request);
        PlaylistParameters params = new PlaylistParameters(prop);

        // this is what's in their current playlist
        PlaylistStatus status = new PlaylistStatus(prop.userID());
        status.init(out);


        if (prop.debug())
            out.print(status.toString());

        if (status.isStale())
        {

            ServletOutputStream outStream = null;

            params = new PlaylistParameters(status);
            if (prop.debug())
            {
                outStream = out;
                out.println("refreshing playlist with parameters: " + params.toString());
                out.flush();
            }

            PlaylistGenerator gen = new PlaylistGenerator(params, songCache, outStream);
            Playlist playlist   = gen.create(!prop.dontsave(), status.playlist);

        }
        else
        {
            out.println("No need to refresh playlist now");
        }

        out.close();
    }
    public void doGet (
        HttpServletRequest      request,
        HttpServletResponse     response
        ) throws ServletException, IOException {
        try
        {

            //Util.debug("PlaylistGeneratorServlet recieved a Get");
            // prevent caching
            response.setHeader("Pragma", "no-cache");
            response.setHeader("Cache-control", "no-cache");
            response.setHeader("Expires", "0");
```

```
                        // figure out what we need to do
                        String actionStr = request.getParameter("action");
                        if (actionStr == null)
                                actionStr = new String("generate");
                        if (actionStr.equals("refresh"))
                        {
                                refreshPlaylist(request, response);
                        }
                        else if (actionStr.equals("cachestatus"))
                        {
                                ServletOutputStream out = response.getOutputStream();
                                response.setContentType("text/plain");
                                songCache.ratingsCache.status(out, request.getParameter("detail") != null);

                                out.close();
                        }
                        else  //default action
                        {
                                generatePlaylist(request, response);
                        }
                }
                catch (Throwable e)
                {
                        System.err.println(new Date().toString() + " Caught an exception in doGet: " +
e.toString());
                        e.printStackTrace();
                }
        }
        public void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException
        {
                Util.debug("PlaylistGeneratorServlet recieved a Post");
                try
                {
                        String user_agent=req.getHeader("USER_AGENT");

                        if (user_agent.equals(com.launch.misc.constants.PLAYLIST_SERVER))
                        {
                                // need to generate play list and return it
                                GeneratorParameters prop = new GeneratorParameters(req);
                                PlaylistParameters params = new PlaylistParameters(prop);
                                PlaylistGenerator gen = new PlaylistGenerator(params, songCache, null);
                                Playlist playlist = gen.create(true, null);

                                Playlist2 playlist2 = playlist.toPlaylist2();

                                ObjectOutputStream oos=new ObjectOutputStream(resp.getOutputStream());
                                oos.writeObject(playlist2);
                                oos.flush();
                                oos.close();
                        }
                        else if (user_agent.equals(com.launch.misc.constants.RATING_WIDGET))
                        {
                                // need to update cache with new info

                                int data_size=req.getContentLength();
                                byte b[]=new byte[data_size];
                                req.getInputStream().read(b,0,data_size);
                                Vector v=(Vector)(new ObjectInputStream(new
ByteArrayInputStream(b))).readObject();
                                Util.debug("received a list of changed ratings " + v);
                                // need to tell cache of these changes
```

```
                                        Enumeration e=v.elements();
                                        while (e.hasMoreElements())
                                        {
250
        songCache.ratingsCache.putIntoCache((CachedRating)e.nextElement());
                                        }
                                }
                                else
                                {
255                                     System.err.println("PlaylistGeneratorServlet received a post from an unknown
person : " + user_agent);
                                }
                        }
                        catch (Throwable t)
260                     {
                                t.printStackTrace();
                        }
                }
                /**
265              * Initialization method -
                 *
                 */
                public void init (ServletConfig config) throws ServletException
                {
270                     super.init(config);
                        songCache = new SongInfoCache(null);
                        // start the updater thread
                        cacheUpdater = new SongInfoCacheUpdater(this);
                        cacheUpdater.setPriority(Thread.MIN_PRIORITY);
275                     cacheUpdater.start();

                        songCache.ratingsCache = new RatingsCache();

                }
280             /**
                 * Destroy method -
                 * get rid of the api
                 * servlets "should have" a destroy method for garbage collection
                 */
285             public void destroy()
                {
                        cacheUpdater.stop();
                        cacheUpdater = null;
                        songCache    = null;
290             }
}
```

PlaylistMaker

```
package com.launch.PlaylistGenerator;
import javax.servlet.ServletOutputStream;
/**
 * this is the dumb class for ASP
 */
public class PlaylistMaker
{
        public PlaylistGenerator generator;
        public Playlist playlist;

        public PlaylistMaker()
        {
                generator = new PlaylistGenerator();
        }

        public void init(int userID, int djID, short mediaType, int moodID, int playlistID)
        {
                // generator.init(userID, djID, moodID);
        }
        public int make()
        {
                playlist = generator.create(false, null);

                return playlist.ID;
        }
        public int makeAndSave()
        {

                playlist = generator.create(true, null);
                return playlist.ID;

        }

        public void toMatrix(ServletOutputStream out, int displayType)
        {
                generator.toMatrix(out, displayType);
        }

        public String toASX()
        {
                return playlist.toASX();
        }

}
```

PlaylistParameters

```
package com.launch.PlaylistGenerator;
public class PlaylistParameters
{

        int userID;
        int djID;
        int playlistSize = Constants.DEFAULT_PLAYLIST_SIZE;
        int moodID;
        Bandwidth speed = new Bandwidth();
        MediaFormat format = new MediaFormat();
        public PlaylistParameters(int userID)
        {
                this.userID = djID = userID;
        }

        public PlaylistParameters(int userID, Bandwidth speed, int moodID)
        {
                this(userID);

                if (speed != null)
                {
                        this.speed = speed;
                }

                this.moodID = moodID;
        }
        public PlaylistParameters(int userID, Bandwidth speed, int moodID, int djID)
        {
                this(userID, speed, moodID);

                if (djID > 0)
                        this.djID = djID;
        }
        public PlaylistParameters(PlaylistStatus status)
        {
                this(status.userID, status.speed, status.moodID, status.djID);
        }

        public PlaylistParameters(GeneratorParameters prop)
        {
                this(prop.userID(), prop.speed(), prop.moodID(), prop.djID());
        }

        public String toString()
        {
                return "userID=" + userID + ", "
                        + "bandwidth=" + speed.toString() + ", "
                        + "moodID=" + moodID + ", "
                        + "djID=" + djID;
        }

}
```

PlaylistStatus

```
package com.launch.PlaylistGenerator;
import java.util.Date;
import javax.servlet.ServletOutputStream;
public class PlaylistStatus
{
        int userID, newRatingsCount, moodID, djID, songsRemaining;
        short mediaType;

        Date lastPlaylist = new Date();

        MediaFormat format;
        Bandwidth speed;

        Date dbDate = new Date();

        public SimplePlaylist playlist;

        public PlaylistStatus(int userID)
        {
                format = new MediaFormat(MediaFormat.WINDOWSMEDIA);
                this.userID = userID;
        }

        public String toString()
        {
                return "Playlist status for userID " + userID + ":" + Util.newLine
                        + " newRatingsCount: " + newRatingsCount + Util.newLine
                        + " moodID: " + moodID + Util.newLine
                        + " djID: " + djID + Util.newLine
                        + " songsRemaining: " + songsRemaining + Util.newLine
                        + " mediaType: " + mediaType + Util.newLine;
        }

        public void init(ServletOutputStream out)
        {
                try
                {
                        DBConnection conn = new DBConnection();

                        DBResultSet rs = conn.executeSQL("exec sp_lcGetPlaylistInfoForUser_xsxx " +
userID);

                        while (!rs.getBOF() && !rs.getEOF())
                        {
                                newRatingsCount = rs.getInt("newRatingsCount");
                                lastPlaylist    = rs.getTimestamp("lastPlaylist");
                                dbDate          = rs.getTimestamp("dbDate");
                                playlist        = SimplePlaylist.fromBytes(rs.getBytes("playlist"));
                                rs.next();
                        }

                        if (playlist != null)
                        {
                                songsRemaining = playlist.songs.size();
                                moodID         = playlist.moodID;
                                djID           = playlist.djID;
                                mediaType      = playlist.mediaType;
                                speed          = Media.typeToBandwidth(mediaType);
                        }
```

```
                conn.close();
            }
            catch (DBException oops)
            {
                Util.out(out, "DBException in PlaylistStatus.init: " + oops.toString());
            }

        }

        public void resetDates()
        {
            if (playlist == null)
                return;

            Util.debug(new Date().toString() + " Playlist OK, just resetting dates for userID " + userID);
            playlist.resetDates(dbDate);
            playlist.save(userID);
        }

        public boolean isStale()
        {

            double oneWeek = Util.MILLISECONDS_IN_SECOND *
                             Util.SECONDS_IN_MINUTE *
                             Util.MINUTES_IN_HOUR *
                             Util.HOURS_IN_DAY *
                             Util.DAYS_IN_WEEK;

            if (songsRemaining <= Constants.REFRESH_AT_SONGS_LEFT)
                return true;

            // if you're listening to someone else's station, your new ratings
            // won't make a difference
            if (newRatingsCount >= Constants.REFRESH_AT_NEW_RATINGS_COUNT && userID ==
djID)
                return true;

            if (new Date().getTime() - lastPlaylist.getTime() > oneWeek)
                return true;

            return false;
        }

        /*
        public void flushPlaylist(ServletOutputStream out)
        {
            try
            {
                DBConnection conn = new DBConnection();
                DBResultSet rs = conn.executeSQL("exec sp_lcFlushPlaylist_xxud " + userID);
                conn.close();
            }
            catch (DBException oops)
            {
                Util.out(out, "DBException in PlaylistStatus::flushPlaylist: " + oops.toString());
            }
        }

        public void deletePlaylist(ServletOutputStream out)
        {
            try
```

```
                {
                        DBConnection conn = new DBConnection();
                        DBResultSet rs = conn.executeSQL("exec sp_lcDeletePlaylist_xxud " + userID);
                        conn.close();
                }
                catch (DBException oops)
                {
                        Util.out(out, "DBException in PlaylistStatus::deletePlaylist: " + oops.toString());
                }
        }


        public void resetClipSchedule()
        {
                try
                {
                        DBConnection conn = new DBConnection();
                        DBResultSet rs = conn.executeSQL("exec sp_lcResetClipSchedule_xxux " + userID);
                        conn.close();
                }
                catch (DBException oops)
                {
                        Util.debug("DBException in PlaylistStatus::resetDates: " + oops.toString());
                }
        }
        */
}
PlaylistStatus.java        Page 3 of 3        11/05/99 1:24 PM
```

PopularSongs

```
package com.launch.PlaylistGenerator;
import java.util.Vector;
import java.util.Hashtable;
import java.util.Enumeration;
public class PopularSongs
{
        private Hashtable byMedia;

        public SongList get(short mediaType)
        {
                return (SongList) byMedia.get(new Short(mediaType));
        }

        public PopularSongs(Hashtable songs, Hashtable mediaTypes)
        {
                byMedia = new Hashtable(1);

                // make a list of all songs and sort them
                SongList all = new SongList(songs);
                all.sort();

                // create each of the song lists
                for (Enumeration e = mediaTypes.keys(); e.hasMoreElements();)
                {
                        Short mediaType = new Short(((Integer) e.nextElement()).shortValue());
                        byMedia.put(mediaType, new SongList());
                }

                SongInfo info;
                Media track;
                SongList list;

                // put each into a separate list for each mediaType
                for (int i = 0; i < all.size(); i++)
                {
                        info = all.elementAt(i);

                        for (int j = 0; j < info.media.size(); j++)
                        {
                                track = info.media.typeAt(j);
                                list = ((SongList) byMedia.get(new Short(track.mediaType)));
                                list.addElement(info);
                        }

                }

                // truncate each list to the top 1000 most popular songs
                for (Enumeration e = mediaTypes.keys(); e.hasMoreElements();)
                {
                        Short mediaType = new Short(((Integer) e.nextElement()).shortValue());
                        list = (SongList) byMedia.get(mediaType);
                        list.setSize(1000);
                }


        }
}
```

Population

```
package com.launch.PlaylistGenerator;
import java.util.Enumeration;
import java.util.Date;
import java.text.SimpleDateFormat;
import java.util.Vector;
import java.util.Hashtable;
import javax.servlet.ServletOutputStream;
import java.text.DateFormat;
public class Population
{

        /*
        private int readers = 0;
        private int writersWaiting = 0;
        private boolean writing = false;
        */

        private boolean haveTitles = false;
        public boolean ordered = false;

        public SongGroup explicit;
        public SongGroup implicit;
        public SongGroup unrated;

        private Hashtable hash;

        public IntHash artistCounts;
        public IntHash albumCounts;

        public Population()
        {
                explicit    = new SongGroup();
                implicit    = new SongGroup();
                unrated     = new SongGroup();
                artistCounts = new IntHash();
                albumCounts  = new IntHash();
                hash        = new Hashtable();
        }

        /*

        public synchronized void addReader()
        {
                ++readers;
        }

        public synchronized void removeReader()
        {
                --readers;
                if (readers == 0)
                {
                        notifyAll();
                }
        }
        public synchronized void requestWrite()
        {
                ++writersWaiting;
        }
```

```
        public synchronized void finishWrite()
        {
                --writersWaiting;
                if (writersWaiting == 0)
                {
                        notifyAll();
                }
        }

        */

        // returns this song if it's valid for adding data, null otherwise

        public synchronized Song initSong(int songID, short type)
        {
                if (type <= 0)
                        return null;

                boolean result = true;
                /*
                requestWrite();

                while (readers > 0)
                {
                        try { wait(); }
                        catch (InterruptedException e) {}
                }
                writing = true;
                */

                Song song = get(songID);

                if (song == null)
                {
                        song = new Song(songID, type);
                        put(songID, song);

                        // if it's excluded, it's not valid for modifying
                        if (type == Song.EXCLUDED)
                                result = false;
                }
                else
                {
                        result = song.setType(type);
                }

                if (result)
                        return song;

//              writing = false;
//              finishWrite();
                return null;
        }

        public synchronized SongData initSongGetData(int songID, short type)
        {
                Song aSong = initSong(songID, type);

                if (aSong == null)
                        return null;

                return aSong.getData();
```

```
    }

    public synchronized SongData getSongData(int songID)
    {
        return getSongData(new Integer(songID));
    }

    public synchronized SongData getSongData(Integer songID)
    {
        Song s = get(songID);

        if (s == null)
            return null;
        return s.getData();
    }


    public synchronized SongData getSongData(int songID, short type)
    {
        SongData result = null;

        /*
        synchronized (this)
        {
            while (writersWaiting > 0)
            {
                try { wait(); }
                catch (InterruptedException e) { }
            }
            addReader();
        }
        */

        Song song = get(songID);

        // there's no song for that ID; Did you call initSong?
        if (song != null && type >= song.getType())
            result = song.getData();
//      removeReader();

        return result;
    }

    public synchronized Song get(int songID)
    {
        return get(new Integer(songID));
    }

    public synchronized Song get(Integer songID)
    {
        return (Song) hash.get(songID);
    }


    public synchronized Song remove(int songID)
    {
        return remove(new Integer(songID));
    }

    public synchronized Song remove(Integer songID)
    {
        return (Song) hash.remove(songID);
```

```
    }

    private synchronized Song put(int songID, Song song)
    {
        return (Song) hash.put(new Integer(songID), song);
    }

    private int available()
    {
        int i = 0;

        for (Enumeration e = hash.keys(); e.hasMoreElements() ;) {
            Song song = get((Integer) e.nextElement());

            if (song.type != Song.EXCLUDED)
            {
                i++;
            }
        }
        return i;
    }

    public Enumeration keys()
    {
        return hash.keys();
    }

    public void order()
    {
        createVectors();
        sortVectors();
    }

    public int excludedCount()
    {
        int result = 0;

        for (Enumeration e = hash.keys(); e.hasMoreElements() ;) {
            Song song = get(((Integer) e.nextElement()).intValue());
            if (song.type == Song.EXCLUDED)
            {
                result++;
            }
        }

        return result;
    }

    public boolean isEligible(int songID, int artistID, int albumID)
    {

        Song song = get(songID);

        if (song != null && song.type == Song.EXCLUDED)
            return false;

        if ((artistCounts.get(artistID) < Constants.RIAA_MAX_SONGS_BY_ARTIST)
            && (albumCounts.get(albumID) < Constants.RIAA_MAX_SONGS_FROM_ALBUM))
            return true;

        return false;
    }
```

```
        public void createVectors()
250     {

                explicit.removeAllElements();
                implicit.removeAllElements();
                unrated.removeAllElements();
255
                for (Enumeration e = hash.keys(); e.hasMoreElements();) {
    //                  Util.debug("interation " + i);
                        Song mySong = get((Integer) e.nextElement());

260                     if (mySong != null)
                        {
                                SongData data = mySong.getData();

                                if (mySong.type == Song.EXPLICIT)
265                                     explicit.addElement(data);
                                else if (mySong.type == Song.IMPLICIT)
                                        implicit.addElement(data);
                                else if (mySong.type != Song.EXCLUDED)
                                        unrated.addElement(data);
270                     }
                }
        }

        public void importPopular(SongList abunch, PlayDates lastPlayed, boolean playBadWords)
275     {
                SongInfo info;
                SongData data;
                Song ditty;
                int added = 0;
280
                unrated.setSize(0);

                long now = new Date().getTime();

285             long lastThreeHours = Util.MILLISECONDS_IN_SECOND *
                                                        Util.SECONDS_IN_MINUTE *
                                                        Util.MINUTES_IN_HOUR *
                                                        3;

290             long playedTime = 0;

                Date playedAt;

                for (int i = 0; i < abunch.size(); i++)
295             {
                        info = abunch.elementAt(i);
                        playedAt = lastPlayed.get(info.songID);

                        // don't play songs twice within 3 hours
300                     if (playedAt == null || (now - playedAt.getTime()) > lastThreeHours)
                        {

                                if (playBadWords || !info.hasExplicitLyrics())
                                {
305
                                        data = initSongGetData(info.songID, Song.UNRATED);

                                        if (data != null)
```

```
                                        {
                                            data.setInfo(info);
                                            unrated.addElement(data);
                                            added++;
                                        }
                                    }
                                }
                            }

                            Util.debug("import popular added " + added + " songs");
                        }

                        public void sortVectors()
                        {
                            sort(explicit, 0, explicit.size() - 1);
                            sort(implicit, 0, implicit.size() - 1);
                            sort(unrated,  0, unrated.size()  - 1);

                //          Util.debug("after sorting, ratedVector is: " + ratedVector.toString());
                //          Util.debug("after sorting, unratedVector is: " + unratedVector.toString());
                            ordered = true;
                        }

                        public void sort(Vector a)
                        {
                            sort(a, 0, a.size() - 1);
                        }

                        private void sort(Vector a, int from, int to)
                        {

                            // quicksort

                            // If there is nothing to sort, return

                            if ((a == null) || (a.size() < 2)) return;

                            int i = from, j = to;
                            SongData center = (SongData) a.elementAt((from + to) / 2);

                            do {
                                while((i < to) && (center.score < ((SongData) a.elementAt(i)).score)) i++;
                                while((j > from) && (center.score > ((SongData) a.elementAt(j)).score)) j--;
                                if (i < j) {
                                    SongData temp = (SongData) a.elementAt(i);
                                    a.setElementAt(a.elementAt(j), i);
                                    a.setElementAt(temp, j);   // swap elements
                                }

                                if (i <= j) { i++; j--; }
                            } while(i <= j);

                            if (from < j) sort(a, from, j); // recursively sort the rest
                            if (i < to) sort(a, i, to);

                        }

                        public String toString()
                        {

                            String result = "";
```

```
        for (Enumeration e = hash.keys(); e.hasMoreElements() ;) {

                int songID = ((Integer) e.nextElement()).intValue();
                Song song = get(songID);

                result = result.concat("songID " + songID
                                                + " = " + song.toString()
                                                + Util.newLine);

        }

        return result;
}

public String sourceCount()
{
        IntHash counts = new IntHash();
        String explicitList = "";

        for (Enumeration e = hash.keys(); e.hasMoreElements() ;) {

                Song song = get(((Integer) e.nextElement()).intValue());

                if (song.getType() == Song.EXPLICIT)
                {
                        explicitList = explicitList.concat(song.songID + ", ");

                }

                counts.increment(song.type);

        }

        return "counts: EXPLICIT = " + counts.get(Song.EXPLICIT)
                        + " (" + explicitList + ") "
                        + " IMPLICIT = " + counts.get(Song.IMPLICIT)
                        + " EXCLUDED = " + counts.get(Song.EXCLUDED);

}

public void toMatrix(ServletOutputStream out, int songType, int displayType)
{
        String delim = "";
        String prefix = "";
        String suffix = "";
        String rowPrefix = "";
        String rowSuffix = "";
        String bold = "";
        String unbold = "";

        if (displayType == Util.DISPLAY_HTML)
        {
                delim = "</TD><TD>";
                prefix = "<TABLE CELLPADDING=1 CELLSPACING=0>";
                suffix = "</TABLE>";
                rowPrefix = "<TR><TD>";
                rowSuffix = "</TD></TR>";
                bold = "<B><FONT SIZE=\"-1\">";
                unbold = "</FONT></B>";
        }
        else
        {
```

```
                delim = "\t";
            }

            Util.out(out, prefix);

            String header = Util.newLine + rowPrefix + bold
                                + Util.join(unbold + delim + bold, SongData.namesArray())
                                + unbold + rowSuffix;


            Vector v = null;

            if (songType == Song.EXPLICIT)
                v = explicit;
            else if (songType == Song.IMPLICIT)
                v = implicit;
            else
                v = unrated;

            if (v != null)
            {

                for (int i = 0; i < v.size(); i++) {

                    SongData data = (SongData) v.elementAt(i);

                    if (i % 40 == 0)
                        Util.out(out, header);

                    Util.out(out, data.toDisplayString(displayType, (i + 1)));


                }
            }

            Util.out(out, suffix);
        }

}
```

Rating

```
package com.launch.PlaylistGenerator;
public class Rating
{
        protected short rating;
        protected boolean set = false;
        public Rating()
        {

        }

        /**
         * create one with a default
         */

        public Rating(short defaultRating)
        {
                rating = defaultRating;
        }

        public boolean isSet()
        {
                return set;
        }

        public void set(short newRating)
        {
                rating = newRating;
                set    = true;
        }

        public short get()
        {
                return rating;
        }

        public String toString()
        {
                if (!set)
                        return rating + "(Not Set)";
                else
                        return "" + rating;
        }

}
```

Rating.java Page 1 of 1 11/05/99 1:28 PM

RatingsCache

```
package com.launch.PlaylistGenerator;
import java.util.*;
import javax.servlet.ServletOutputStream;
import java.io.IOException;
public final class RatingsCache implements GetRatingsCacheUsersInterface, Constants
{
        /**
         * This Hashtable will be of the form
         * (Integer userID, Hashtable CachedRating objects), if the Data in
         * the cache is invalid the entry will be of the form
         * (Integer userID, INVALID_DATA)
         * <br>
         * The Hashtable of CachedRating objects is of the form (Integer itemID, CachedRating)
         **/
        private Hashtable ratingsList = new Hashtable(1);
        private GetRatingsCacheUsers gtu;

        private FrequencyCounter freq_counter = new
FrequencyCounter(RATINGS_CACHE_INITIAL_SIZE);

        private Date lastUpdated = new Date();
        private Date lastReset   = new Date();


    //-------------------------------------------------------
        public RatingsCache()
        {
                gtu = new GetRatingsCacheUsers(this);
                // the following line is for testing purposes only, rem it out otherwise.
//              gtu.SLEEP_TIME=5*60*1000;
                gtu.start();
        }
        /**
         * This method will get a list of rating for the given userids
         * @param userid an array of ints representing userids, each entry should be a valid userID, do not
pad with zeros.
         * @return a Vector of CachedRating objects
         **/
        public final Vector getRatings(Vector users)
        {
                //----------
                // algorithm
                //----------
                // check for userid in hashtable
                // if found add to vector of ratings
                // else build list of unfound things
                //     get list of unfound things from database
                Vector allRatings    = new Vector();
                Integer userID;
                Hashtable ratingProfile;
                Vector nonCachedUsers = new Vector(users.size());
                Date startDate       = new Date();
                Enumeration e        = users.elements();
                while (e.hasMoreElements())
                {
                        userID       = (Integer) e.nextElement();
                        ratingProfile  = (Hashtable) ratingsList.get(userID);
                        if (ratingProfile == null)
                        {
```

```
                                        Util.debug("RatingsCache MISS on user " + userID);
                                        nonCachedUsers.addElement(userID);
                                }
                                else
                                {

//                                      benchmark_date1 = new Date();
                                        Util.debug("RatingsCache HIT on user " + userID);
                                        appendToVector(allRatings, ratingProfile.elements());
//                                      Util.printElapsedTime("Get from cache, " + temp_hash.size() + "
entries", benchmark_date1);
                                }
                                freq_counter.incrementValue(userID);
                        }
                        if (nonCachedUsers.size() > 0)
                        {
                                MergeVectors(allRatings, getRatingsFromDatabase(nonCachedUsers));
                        }
                        Util.printElapsedTime(Thread.currentThread().getName() + ", got " + allRatings.size() +
" ratings ", startDate);
                        return allRatings;
                }
                public final void updateCachedUsers(Vector v)
                {
                        setCachedUserIDs(v);
                }
                public Hashtable getMostFrequentlyUsedUsers(int i)
                {
                        Hashtable h = freq_counter.getLargest(i);
                        Vector    v = new Vector(h.size());

                        // when we do this, also refresh the cache
                        // to clean out any lingering data corruption

                        Util.debug(new Date().toString() + " Resetting ratings cache");

                        // clear the users in the cache
                        setCachedUserIDs(v);

                        lastReset = new Date();

                        // put user hash into vector
                        appendToVector(v, h.keys());

                        // get all the ratings
                        setCachedUserIDs(v);

                        return h;
                }
                /**
                 *
                 **/
                public final void setCachedUserIDs(Vector userIDs)
                {
                        lastUpdated = new Date();

                        Vector cachedUsers = (Vector) userIDs.clone();
                        Date benchmark_date = new Date();
                        if (cachedUsers.size() <= 0)
                        {
                                ratingsList = new Hashtable(1);
                                Util.debug("setCachedUserIDs: no users passed");
```

```
                    return;
                }
Enumeration e = ratingsList.keys();
Integer userID;
// find the differences between the users already in the cache
// and the new list of users
// leave that result in cachedUsers

// interate through each user in the current cache
while (e.hasMoreElements())
{
    userID = (Integer) e.nextElement();
    // are they in the new list?
    if (cachedUsers.contains(userID))
    {
        // cool, just remove them from the new list
        cachedUsers.removeElement(userID);
    }
    else
    {
        // they've been removed
        ratingsList.remove(userID);
    }
}
Vector newRatings = new Vector();

// get all the ratings for the new cached users

if (cachedUsers.size() > 0)
{
    newRatings = getRatingsFromDatabase(cachedUsers);
    e = newRatings.elements();
    while (e.hasMoreElements())
    {
        putIntoCache((CachedRating) e.nextElement());
    }
}
else
{
    Util.debug(new Date().toString() + " setCachedUserIDs: no new users in
cache");
}
Util.printElapsedTime("refreshed cached users and loaded " + newRatings.size() + "
entries", benchmark_date);
    }
    /**
     *
     **/
    private final Vector getRatingsFromDatabase(Vector userIDs)
    {
        //----------
        // algorithm
        //----------
        // query database for info
        // build vector from resultsets.
            Vector results = new Vector(RATINGS_CACHE_INITIAL_SIZE);
            Date benchmark_date = new Date();
        //--- get item rating ---
            GetItemRatingsFromDB itemRatings = new GetItemRatingsFromDB(userIDs,
results);
        //--- get song rating ---
            GetSongRatingsFromDB songRatings = new GetSongRatingsFromDB(userIDs,
```

```
results);
                                songRatings.start();
                                itemRatings.start();
                        //--- must wait for the two threads to finish ---
                                try
                                {
                                        itemRatings.join();
                                        songRatings.join();
                                }
                                catch (InterruptedException e)
                                {
                                        System.err.println("PlaylistCache: interrupted waiting for ratings, I'm not cleanning up...");
                                }
                        //--- done getting just return values ---
                                Util.printElapsedTime("GetRatingsFromDatabase, " + results.size() + " entries", benchmark_date);
                                return results;
                }
                /**
                 * appends the contents of vector2 into vector1
                 **/
                private static final void MergeVectors(Vector vector1, Vector vector2)
                {
                        vector1.ensureCapacity(vector1.size() + vector2.size());
                        Enumeration e = vector2.elements();
                        while (e.hasMoreElements())
                        {
                                vector1.addElement(e.nextElement());
                        }
                }
                public static final void appendToVector(Vector v, Enumeration e)
                {
                        while (e.hasMoreElements())
                        {
                                v.addElement(e.nextElement());
                        }
                }
                public static final String GetVectorAsCommaDelimitedList(Vector v)
                {
                        if (v==null) return("");
                        String s=v.toString();
                        int vector_length=s.length();
                        if (vector_length >= 3)
                        {
                                return(s.substring(1,vector_length-1));
                        }
                        else
                        {
                                return("");
                        }
                }
                /**
                 * This method adds the value to the hashtable pointed to by the key, if the key does not exist yet it will create the first entry and the Hashtable
                 **/

                public final void putIntoCache(CachedRating value)
                {
                        RatingsProfile profile = null;
                        Integer userID = new Integer(value.userID);
                        // this could be more efficient if we inserted all the ratings for a particular user all at once
```

```
                        if (ratingsList.containsKey(userID))
                        {
                                profile = (RatingsProfile) ratingsList.get(userID);
                        }
                        else
                        {
                                profile = new RatingsProfile(RATINGS_CACHE_INITIAL_SIZE);
                                ratingsList.put(userID, profile);
                        }
                        if (value.rating < 0)
                        {
                                // unrate
                                profile.remove(value.hashKey());
                        }
                        else
                        {
                                profile.put(value.hashKey(), value);
                        }
                }
                public final String toString()
                {
                        return ratingsList.toString();
                }

                public final String userList()
                {
                        String result = "";

                        Enumeration e = ratingsList.keys();
                        Integer userID;

                        while (e.hasMoreElements())
                        {
                                userID = (Integer) e.nextElement();
                                result = result.concat(userID + ", ");
                        }

                        return result;
                }

                public final void status(ServletOutputStream out, boolean detail) throws IOException
                {
                        out.print("RatingsCache has " + ratingsList.size() + " users" + Util.newLine
                                                                        + "Last Updated at " +
lastUpdated.toString() + Util.newLine
                                                                        + "Last Reset at " + lastReset.toString() +
Util.newLine
                                                                        + "UserList is " + userList() +
Util.newLine);

                        Enumeration e = ratingsList.keys();
                        Integer userID;
                        RatingsProfile profile;

                        while (e.hasMoreElements())
                        {
                                userID = (Integer) e.nextElement();
                                out.print(Util.newLine + "Profile for userID " + userID + ":" + Util.newLine);

                                profile = (RatingsProfile) ratingsList.get(userID);

                                if (profile == null)
```

```
                {
310                     out.print("NULL!" + Util.newLine);
                }
                else
                {
                        out.print(Util.newLine + profile.count(Constants.ITEM_TYPE_SONG)
315     + " songs, "
                                                                                +
        profile.count(Constants.ITEM_TYPE_ALBUM) + " albums, "
                                                                                +
        profile.count(Constants.ITEM_TYPE_ARTIST) + " artists, "
320                                                                             + profile.count((byte) 0)
        + " total" + Util.newLine);

                        if (detail)
                                out.print(profile.toString());
325             }
        }
    }
}
```

RatingsProfile

```
package com.launch.PlaylistGenerator;
import java.util.Hashtable;
import java.util.Enumeration;
public class RatingsProfile extends Hashtable
{
        public RatingsProfile(int capacity)
        {
                super(capacity);
        }

        public int count(byte type)
        {
                int count = 0;

                if (type <= 0)
                        return size();
                else
                {
                        Enumeration e = keys();

                        String key;

                        CachedRating rating;

                        while (e.hasMoreElements())
                        {
                                key = (String) e.nextElement();

                                rating = get(key);

                                if (rating.type == type)
                                        count++;
                        }
                }

                return count;
        }

        public CachedRating get(String key)
        {
                return (CachedRating) super.get(key);
        }

        public String toString()
        {
                String result = "";
                Enumeration e = keys();

                while (e.hasMoreElements())
                {
                        result = result.concat((get((String) e.nextElement())).toString());
                }

                return result;
        }
}
```

RatingWidgetServlet

```
package com.launch.PlaylistGenerator;
import java.util.*;
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;
/**
*-------------------------------------------------------------------
*
* RatingWidgetServlet.java 7/8/99
* Initial Servlet for ratings Widget
* Copyright (c) 1999 LAUNCH Media, Inc.
* @author Jon Heiner
*-------------------------------------------------------------------
*/
public class RatingWidgetServlet extends HttpServlet implements GetRatingsCacheUsersInterface,
GetPlaylistServersInterface, Runnable
{
        private Vector cachedUsers = new Vector(1);
        private GetRatingsCacheUsers gtu;
        private Vector playlistServers = new Vector(1);
        private GetPlaylistServers gps;
        /** This vector contains CachedRating objects */
        private Vector dirtyRatings = new Vector(Constants.RATING_UPDATE_LIST_INITIAL_SIZE);
        private Thread myThread;
    //-------------------
        /**
         * Handle requests...
         */
        public void doGet (
                HttpServletRequest      request,
                HttpServletResponse     response
                ) throws ServletException, IOException
        {
                String sEvent;
                String sRater;
                String sRatee;
                int iRateeType;
                String sRating;
                int raterID = 0;

                // get parameters
                sEvent = request.getParameter("action");
                // get stream for output
                ServletOutputStream out;
                response.setContentType("text/plain");
                response.setHeader("Pragma", "no-cache");
                response.setHeader("Cache-control", "no-cache");
                response.setHeader("Expires", "0");

                out = response.getOutputStream();
                try
                {
                        DBConnection conn = new DBConnection();
                        if (sEvent.equals("INIT"))
                        {
                                sRater = request.getParameter("rater");
                                sRatee = request.getParameter("ratee");
                                iRateeType = Integer.parseInt( request.getParameter("ratee_type") );
```

```
                                        int rating      = -1; // not rated
                                        boolean implicit = false;
                                        String sql = "";
                                        // SONG case
                                        if (iRateeType == Constants.ITEM_TYPE_SONG)
                                        {
                                                sql = "exec sp_lcGetSongInfoSummary_xsxx "
                        + sRater + ","
                        + sRatee;
                                        }
                                        else if (iRateeType == Constants.ITEM_TYPE_ALBUM)
                                        {
                                                sql = "exec sp_lcGetArtistOrAlbumRating_xsxx "
                        + sRatee + ","
                        + sRater;
                                        }
                                        else
                                        {
                                                sql = "exec sp_lcGetArtistOrAlbumRating_xsxx "
                        + sRatee + ","
                        + sRater;
                                        }

                                        DBResultSet rs = conn.executeSQL(sql);
                                        if (!rs.getBOF() && !rs.getEOF())
                                                rating = rs.getInt("rating");
                                        out.println("rating_value=" + rating +
"&Implicit_indicator=not_implicit");
                                }
                                else if (sEvent.equals("RATING_EVENT"))
                                {
                                        /* Do update to LaunchCast Ratings Database */
                                        sRater    = request.getParameter("rater");

                                        try
                                        {
                                                raterID = Integer.parseInt(sRater);
                                        }
                                        catch (NumberFormatException e)
                                        {
                                                throw new Exception("RatingWidgetServlet: rating received
for invalid user: " + sRater);
                                        }

                                        if (raterID <= 0)
                                        {
                                                throw new Exception("RatingWidgetServlet: rating received
for invalid user: " + raterID);
                                        }

                                        sRatee    = request.getParameter("ratee");
                                        iRateeType = Integer.parseInt( request.getParameter("ratee_type") );
                                        sRating   = request.getParameter("rating");
                                        // song case
                                        if (iRateeType == Constants.ITEM_TYPE_SONG)
                                        {
```

```
                                        conn.executeUpdate("exec sp_IcRateSongUser_isux "
                + raterID + ","
                + sRatee + ","
                + sRating, true);
                        }
                        // album case
                        else if (iRateeType == Constants.ITEM_TYPE_ALBUM)
                        {
                                conn.executeUpdate("exec sp_IcRateItemUser_isux "
                + raterID + ","
                + sRatee + ","
                + sRating, true);
                        }
                        // artist case
                        else
                        {
                                conn.executeUpdate("exec sp_IcRateItemUser_isux "
                + raterID + ","
                + sRatee + ","
                + sRating, true);
                        }
                        out.println("confirmation=rating_confirmed");
                        if (cachedUsers.contains(new Integer(raterID)))
                        {
                                CachedRating cr = new CachedRating(raterID,
Integer.parseInt(sRatee), Byte.parseByte(sRating), (byte)iRateeType);
                                dirtyRatings.addElement(cr);
                                Util.debug("Added change to ratings cache update queue : " +
cr);
                        }
                }
                else
                {
                        out.println("error");
                }
                conn.close();
        }
        catch(DBException e) {
                out.println("DBException: " + e.getMessage());
                System.err.println(new Date().toString() + " DBException in
RatingWidgetServlet: " + e.toString());
        }
        catch(Exception e) {
                out.println("Exception raised: " + e);
                System.err.println(new Date().toString() + " Exception in RatingWidgetServlet:
" + e.toString());
        }
        out.close();
    }
    public void init (ServletConfig config)
        throws ServletException {
        super.init(config);
        try {
```

```
                gtu = new GetRatingsCacheUsers(this);
                gps = new GetPlaylistServers(this);
                // the following 2 lines are for testing purposes only, rem them out otherwise.
        //              gtu.SLEEP_TIME=1*20*1000;
        //              gps.SLEEP_TIME=1*20*1000;
                gps.start();
                gtu.start();
                myThread = new Thread(this);
                myThread.start();
            }
            catch (Exception e) { throw new ServletException (); }
        }
        /**
         * Destroy method -
         * get rid of the api
         * servlets "should have" a destroy method for garbage collection
         */
        public void destroy() {
            gps.stop();
            gtu.stop();
        }
    //-----------------------
        public void updateCachedUsers(Vector topUsers)
        {
            cachedUsers = topUsers;
        }
        public void updatePlaylistServers(Vector v)
        {
            playlistServers = v;
        }
        public void run()
        {
            // once every N minutes go update all cached ratings with some new ratings
            Util.debug("RatingWidgetServlet notify playlistgenerators of changed rating - thread
started");
            try
            {
                Vector temp_dirty_ratings;
                Enumeration enum;
                Socket s;
                ByteArrayOutputStream baos;
                ObjectOutputStream oos;
                OutputStream os;
                BufferedWriter bw;
                byte b[];
                String server_ip = null;
                while (dirtyRatings != null)
                {

                    try
                    {

                        if (dirtyRatings.size() > 0)
                        {
                            baos = new ByteArrayOutputStream(1000);
                            oos  = new ObjectOutputStream(baos);
                            temp_dirty_ratings = dirtyRatings;
                            dirtyRatings     = new
Vector(Constants.RATING_UPDATE_LIST_INITIAL_SIZE);
                            // need to send info to cached servers here.
                            oos.writeObject(temp_dirty_ratings);
                            oos.flush();
```

```
                                                b=baos.toByteArray();
                                                enum = playlistServers.elements();
                                                while (enum.hasMoreElements())
                                                {
                                                        try // this nested try / catch is so if one server is down the others get updated too.
                                                        {
        server_ip=(String)enum.nextElement();
                                                                Util.debug(new Date().toString() + " RatingWidgetServlet: Sending changed ratings to : " + server_ip + " this vector : " + temp_dirty_ratings);
                                                                s=new Socket(server_ip, Constants.PORT_NUMBER);
                                                                os=s.getOutputStream();
                                                                bw=new BufferedWriter(new OutputStreamWriter(os));
        bw.write(Constants.POST_HEADER);
                                                                bw.newLine();
        bw.write(com.launch.misc.constants.USER_AGENT + ": " + com.launch.misc.constants.RATING_WIDGET);
                                                                bw.newLine();
                                                                bw.write("Content-length: " + b.length);
                                                                bw.newLine();
                                                                bw.newLine();
                                                                bw.flush();
                                                                os.write(b);
                                                                os.flush();
                                                                os.close();
                                                        }
                                                        catch (Exception e1)
                                                        {
                                                                System.err.println((new Date()).toString() + " Error contacting ratings cache at " + server_ip);
                                                                //e1.printStackTrace();
                                                        }
                                                }
                                        }
                                }
                                catch (Exception e2)
                                {
                                        System.err.println((new Date()).toString() + " Error in RatingWidgetServlet CacheUpdater while loop");
                                        e2.printStackTrace();
                                }
        Thread.sleep(Constants.PROPAGATE_DIRTY_RATING_SLEEP_TIME);
                        }
                }
                catch (Exception e)
                {
                        System.err.println(new Date().toString() + " Fatal Error in RatingWidgetServet updater thread ");
                        e.printStackTrace();
                }
                Util.debug("RatingWidgetServlet notify playlistgenerators of changed rating - thread done");
        }
        public Hashtable getMostFrequentlyUsedUsers(int i)
```

```
                {
310                     return null;
                }
        }
        /* eof */
```

RatingWidgetServlet.java Page 7 of 7 11/05/99 1:35 PM

RecList

```
package com.launch.PlaylistGenerator;
import java.util.Vector;
/**
 * Launch Media, Inc Copyright 1999
 *
 * Recommendation List - class which encapsulates
 * recommendations coming from the net perceptions engine
 *
 * RECOMMENDED USAGE
 * to access values within a RecList object:
 *
 *
 * void someFunction(RecList aRec) {
 *
 *      if ( aRec.setToFirstRec() ) {
 *              do {
 *              System.out.println( aRec.getIdentifier() + " : " + aRec.getPredictedRating() );
 *              } while  aRec.increment() ;
 *      }
 * }
 *
 *
 *
 * The "prediction result" object in net perceptions is NOT
 * persistent so is unusable outside of a carefully controlled
 * environment
 *
 * Further, developers within LAUNCH should not be exposed
 * to Net Perceptions data structures (as they are ugly)
 *
 * file: launchNetP.java
 * @author Jon Heiner
 * @since 7-30-99
 */
public class RecList {
        private final static int kGrowVectorBy = 4;
        private Vector theRecs;
        private int theNumRecs = 0;
        private int theIndex = 1;
        /* Rec -- inner class
         * encapsulates the ID and predicted
         * value for the item in the list;
         * the inner values are made public
         * for convenience; they are exposed
         * to this class, but are not intented
         * to be used outside of this implementation
         */
        public class Rec {
                public int theID;
                public float theValue;
                /* Rec - creation method
                 * the variables should be immutable
                 */
                public Rec(int iID, float fValue) {
                        theValue = fValue;
                        theID = iID;
                }
        }
        /** RecList - creation method
```

```
 * creates an empty rec list, which we will then add
 * Recs to; if you try to pull values from this it will
 * indicate that this is not possible
 */
public RecList() {
        theRecs = new Vector(0, kGrowVectorBy); // create an empty vector
}
/** RecList - creation method w/ args
 * creates a rec list with one element; use the add
 * method to add more values to it
 */
public RecList(int iID, float fValue) {
        theRecs = new Vector(0, kGrowVectorBy); // create an empty vector
        this.add(iID, fValue);
}
/** compact
 * called once the RecList has been created and
 * all items are added
 */
public void compact() {
        theRecs.trimToSize();
}
/** setToFirstRec
 * called to set us to the first rec
 * if this returns false, then there are
 * no recommendations in the list.
 */
public boolean setToFirstRec() {
        theIndex = 0;
        if (theNumRecs > 0) return true;
        return false;
}

/** increment
 * moves the internal pointer to the next item
 * returns false if there are no more Recs in
 * the list.
 */
public boolean increment() {
        theIndex++;
        if (theIndex < theNumRecs) return true;
        return false;
}
/** getidentifier
 * returns the item ID for the current item
 * in the Rec List
 */
public int getIdentifier() {
        return (int) ((Rec) theRecs.elementAt(theIndex)).theID;
}
/** getPredictedRating
 * returns the percentage value which is the
 * predicted value
 */
public float getPredictedRating() {
        return (float) ((Rec) theRecs.elementAt(theIndex)).theValue;
}
/** add
 * adds a new value to the Rec list
 * returns false if the values entered
 * are invalid; (e.g.: iId < 0)
 */
```

```
        public void add(int iID, float fValue) {
                theNumRecs++;
                theRecs.addElement(new Rec(iID, fValue) );
        }

        /** length
         * returns the number of elements in the Rec list
         */
        public int length() {
                return theNumRecs;
        }
        /** createStubRecList
         * used to return "good" bogus values rather
         * than values generated from Net Perceptions
         * useful for testing and stubbing
         */
        public static RecList createStubRecList() {
                RecList aRecList = new RecList(74082, (float) 0.5);
                aRecList.add(116377, (float) 0.6);
                aRecList.add(123312, (float) 0.7);
                aRecList.add(899, (float) 0.8);
                aRecList.add(58075, (float) 0.9);
                return aRecList;
        }
        /** test
         * test class
         */
        public static class Test {

                /*
                public static void main(String [] args) {
                        System.out.println( "debug 0");
                        RecList aRec = createStubRecList();

                        System.out.println( "debug 1");
                        if ( aRec.setToFirstRec() ) {
                        System.out.println( "debug 2");
                                do {
                        System.out.println( "debug 3");
                                System.out.println( aRec.getIdentifier() + " : " + aRec.getPredictedRating() );

                        System.out.println( "debug 4");
                                } while ( aRec.increment() );
                        }
                }
                */
        }
}
```

SaveClips

```
package com.launch.PlaylistGenerator;
import java.util.Vector;
import java.util.Date;
public class SaveClips extends Thread
{
        Vector clips;
        String storedProc;
        int ordinal;
        short mediaType;
        int userID;

        public SaveClips(Vector clips, String storedProc, int ordinal, short mediaType, int userID)
        {
                this.clips     = clips;
                this.storedProc = storedProc;
                this.mediaType  = mediaType;
                this.userID     = userID;
                this.ordinal    = ordinal;
        }
        public void run()
        {
                Date startDate = new Date();
                Thread.currentThread().setName("SaveClips for " + storedProc);

                int rowCount = 0;

                if (clips.size() <= 0)
                        return;

                try
                {
                        DBConnection conn = new DBConnection();
                        String sql = "";

                        Clip aClip;

                        for (int i = 0; i < clips.size(); i++)
                        {
                                aClip = (Clip) clips.elementAt(i);

                                sql = sql.concat(" exec " + storedProc + " "
                                        + ordinal + ", "
                                        + aClip.media.getID(mediaType) + ", "
                                        + mediaType + ", "
                                        + userID);

                                ordinal++;
                                rowCount++;
                        }


                        conn.executeSQL(sql);

                        conn.close();
                }
                catch (DBException oops)
                {
                        Util.debug("DB Exception: " + oops.getMessage());
```

```
                }

                Util.debug(Thread.currentThread().getName() + " saved " + rowCount + " clips");
                Util.printElapsedTime(Thread.currentThread().getName(), startDate);
        }
}
```

SaveClips.java Page 2 of 2 11/05/99 1:25 PM

SavePlaylist

```
package com.launch.PlaylistGenerator;
import java.util.Date;
public class SavePlaylist extends Thread
{
        Playlist list;
        int ordinal, to, from;

        public SavePlaylist(Playlist list, int from, int to, int ordinal)
        {
                this.list    = list;
                this.ordinal = ordinal;
                this.to      = to;
                this.from    = from;
        }
        public void run()
        {
                Date startDate = new Date();
                Thread.currentThread().setName("SavePlaylist (" + from + "-" + to + ")");

                int rowCount = 0;

                try
                {
                        DBConnection conn = new DBConnection();
                        String sql = "";

                        SongData data;
                        short origin;

                        for (int i = from; i < to; i++)
                        {
                                data = (SongData) list.media.elementAt(i);

                                if (list.popularOnly)
                                        origin = (short) SongData.SOURCE_FORCED_POPULAR;
                                else
                                        origin = (short) data.origin();

                                if (data.querySource == SongData.SOURCE_RATED)
                                        origin = (short) data.rating.getSource();

                                //
                                sql = sql.concat(" exec sp_lcSaveMediaPlaylist_ixxd "
                                        + ordinal + ", "
                                        + data.getMediaID(list.mediaType) + ", "
                                        + list.mediaType + ", "
                                        + list.userID + ", "
                                        + data.implicit + ", "
                                        + origin);

                                ordinal++;
                                rowCount++;
                        }
                        conn.executeSQL(sql);
                        conn.close();
                }
                catch (DBException oops)
                {
                        Util.debug("DB Exception: " + oops.getMessage());
```

```
                }

                Util.debug(Thread.currentThread().getName() + " saved " + rowCount + " songs");
                Util.printElapsedTime(Thread.currentThread().getName(), startDate);
        }
    }
```

SavePlaylist.java Page 2 of 2 11/05/99 1:25 PM

SimpleClip

```
package com.launch.PlaylistGenerator;
import java.io.Serializable;
public class SimpleClip implements Serializable
{
        int mediaID;
        int ID;
        byte origin;
        public String toString()
        {
                return "clipID=" + ID + ", mediaID=" + mediaID + ", origin=" + origin;
        }

        /**
         * Contructor for ads, news, tips
         */
        public SimpleClip(int ID, int mediaID)
        {
                this.mediaID = mediaID;
                this.ID  = ID;
        }

        /**
         * Constructor for songs
         */
        public SimpleClip(int ID, int mediaID, byte origin)
        {
                this(ID, mediaID);
                this.origin = origin;
        }
}
```

SimpleClip.java Page 1 of 1 11/05/99 1:32 PM

SimpleClipList

```
package com.launch.PlaylistGenerator;
import java.util.Vector;
public class SimpleClipList extends Vector
{

        public SimpleClipList(int size)
        {
                super(size);
        }

        public SimpleClip pop()
        {
                if (size() > 0)
                {
                        SimpleClip clip = (SimpleClip) elementAt(0);
                        removeElementAt(0);
                        return clip;
                }

                return null;
        }

}
```

SimpleClipList.java Page 1 of 1 11/05/99 1:32 PM

SimplePlaylist

```
package com.launch.PlaylistGenerator;
import java.util.Vector;
import java.io.Serializable;
import java.io.ByteArrayOutputStream;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.ByteArrayInputStream;
import java.util.Date;
public class SimplePlaylist implements Serializable
{
        SimpleClipList news  = new SimpleClipList(10);
        SimpleClipList ads   = new SimpleClipList(10);
        SimpleClipList tips  = new SimpleClipList(10);
        SimpleClipList songs = new SimpleClipList(50);

        Date lastAd;
        Date lastNews;
        Date lastTip;

        short mediaType;
        int moodID;
        int djID;

        public String toString()
        {
                return "ads="   + ads.toString()   + ", " +
                               "news="  + news.toString()  + ", " +
                               "songs=" + songs.toString() + ", " +
                               "tips="  + tips.toString();

        }

        public void resetDates(Date newDate)
        {
                lastAd = lastNews = lastTip = newDate;
        }

        public void save(int userID)
        {

                try
                {
                        DBConnection conn = new DBConnection();
                        save(conn, userID);
                }
                catch (DBException e)
                {
                        System.err.println(new Date().toString() + " DBException in SimplePlaylist:save: " +
e.toString());
                        e.printStackTrace();
                }
        }

        public void save(DBConnection conn, int userID)
        {
                try
                {
                        String sql = "exec sp_lcSavePlaylist_ixxd " + userID + ", ?";
```

```
                        DBPreparedStatement statement = conn.prepareStatement(sql);

                        byte[] b = toByteArray();

                        statement.setBytes(1, toByteArray());

                        statement.executeUpdate();

                }
                catch (DBException e)
                {
                        System.err.println(new Date().toString() + " DBException in SimplePlaylist:save:" +
e.toString());
                }
        }

        public static SimplePlaylist fromBytes(byte[] b)
        {
                if (b == null || b.length <= 0)
                        return null;

                try
                {
                        ByteArrayInputStream bais = new ByteArrayInputStream(b);

                        if (bais == null)
                                return null;

                        ObjectInputStream ois = new ObjectInputStream(bais);

                        if (ois == null)
                                return null;

                        return (SimplePlaylist) ois.readObject();
                }
                catch (Throwable e)
                {
                        System.err.println("Exception in SimplePlaylist:fromBytes:" + e.toString());
                }
                return null;
        }

        public static SimplePlaylist load(DBConnection conn, int userID)
        {
                String sql = "exec sp_lcGetPlaylist_xsxx " + userID;

                try
                {
                        DBResultSet rs = conn.executeSQL(sql);

                        return SimplePlaylist.fromBytes(rs.getBytes("playlist"));
                }
                catch (Throwable e)
                {
                        System.err.println("Exception in SimplePlaylist:load:" + e.toString());
                }
                return null;
        }

        private byte[] toByteArray()
        {
                try
```

```
        {
          ByteArrayOutputStream baos = new ByteArrayOutputStream();
125       ObjectOutputStream oos    = new ObjectOutputStream(baos);
          oos.writeObject(this);
          return baos.toByteArray();
        }
        catch (Throwable t)
130     {
                        System.err.println("toByteArray died: " + t.toString());
          t.printStackTrace();
          return null;
        }
135       }
    }
```

Song

```
package com.launch.PlaylistGenerator;
public class Song
{
        public final static short EXCLUDED = 4;
        public final static short EXPLICIT = 3;
        public final static short IMPLICIT = 2;
        public final static short UNRATED  = 1;
        public final static short ANY      = 0;

        public int songID;
        public short type = ANY;
        private SongData data = null;

        public Song(int songID, short type)
        {
                this.songID = songID;
                setType(type);
        }

        public String toString()
        {
                return "Song " + songID
                        + ", type = "
                        + typeString()
                        + ", data = "
                        + ((data == null) ? "null" : data.toString());
        }

        public String typeString()
        {
                switch (type)
                {
                        case ANY:
                                return "ANY";
                        case EXPLICIT:
                                return "EXPLICIT";
                        case IMPLICIT:
                                return "IMPLICIT";
                        case UNRATED:
                                return "UNRATED";
                        case EXCLUDED:
                                return "EXCLUDED";
                        default:
                                return "UNKNOWN";
                }
        }

        // this should wait for setType
        public SongData getData()
        {
                return data;
        }

        // this should wait for setType
        public short getType()
        {
                return type;
        }
```

```
        // returns whether or not this is suitable for setting SongData
        public boolean setType(short newType)
        {
                short oldType = type;

                if (newType == type)
                        return true;
                else if (newType < type)
                        return false;
                else
                        type = newType;

                // add or delete song data

                if (newType == EXCLUDED)
                {
//                      if (oldType != 0)
//                              Util.debug(Thread.currentThread().getName() + ": deleting data for song " +
songID + ", oldType was " + oldType);
                        data = null;
                }
                else if (oldType == ANY && (newType == EXPLICIT || newType == IMPLICIT || newType ==
UNRATED))
                {
                        data = new SongData(songID);
                }

                return true;
        }
}
```

SongData

```
package com.launch.PlaylistGenerator;
public class SongData
{
        int songID;
        byte querySource;
        public AverageRating djsAverage;
        double score,
                netp,
                implicit,
                confidence,
                lastPlayed,
                bds,
                ratingF,
                djsF,
                netpF,
                commRatingF,
                lastPlayedF,
                bdsF;
        private SongInfo info;
        private Rating djs = new Rating((short) Constants.DEFAULT_DJS_SCORE);
        private byte djSource = SOURCE_DJS;
        public SongRating rating;
        PickStatus status;
        public final static byte SOURCE_RATED        = 1;
        public final static byte SOURCE_IMPLICIT_ALBUM  = 2;
        public final static byte SOURCE_IMPLICIT_ARTIST = 3;
        public final static byte SOURCE_IMPLICIT_SONG   = 4;
        public final static byte SOURCE_DJS          = 5;
        public final static byte SOURCE_DJS_SONG      = 5;
        public final static byte SOURCE_BDS          = 6;
        public final static byte SOURCE_POPULAR       = 7;
        public final static byte SOURCE_RANDOM        = 8;
        public final static byte SOURCE_NETP          = 9;
        public final static byte SOURCE_ALL           = 10;
        public final static byte SOURCE_RECENTLY_PLAYED = 11;
        public final static byte SOURCE_FORCED_POPULAR  = 12;
        public final static byte SOURCE_GENRES                  = 13;
        public final static byte SOURCE_DJS_ALBUM               = 14;
        public final static byte SOURCE_DJS_ARTIST              = 15;
        public final static byte DO_NOTHING     = 0;
        public final static byte MAKE_ME_IMPLICIT = 1;
        public final static byte EXCLUDE_ME     = 2;
        public SongData(int songID)
        {
                lastPlayed  = Constants.DEFAULT_LASTPLAYED_SCORE;
                djsAverage  = new AverageRating((short) Constants.DEFAULT_DJS_SCORE);
                status      = new PickStatus();
                netp                = Constants.DEFAULT_NETP_SCORE;
                this.songID = songID;
                rating      = new SongRating();
        }
        public boolean equals(SongData otherData)
        {
                return (songID == otherData.songID);
        }
        public byte origin()
        {
                double maxValue = 0;
                byte maxSource  = SOURCE_RANDOM;
```

```
        byte ratingSource = 0;
        if (rating.isSet())
                ratingSource = rating.getSource();
                if (info.commRating > maxValue && info.commRating > Constants.POPULAR_THRESHOLD
&& ratingSource != 1)
                {
                        maxValue  = info.commRating;
                        maxSource = SOURCE_POPULAR;
                }
                if (djs.isSet() && djs.get() >= maxValue && djs.get() > 0 && ratingSource != 1)
                {
                        maxValue  = djs.get();
                        maxSource = djSource;
                }
                /*
                if (netP > maxValue)
                {
                        maxValue = netP;
                        maxSource = SOURCE_NETP;
                }
                */
                if (bds > 0 && bds >= maxValue && ratingSource != 1)
                {
                        maxValue  = bds;
                        maxSource = SOURCE_BDS;
                }
                // according to the weight matrix, if there's an explicit rating,
                //that's the only source
                // but let's lie to people because they don't like it when we say
                // we played lowly-rated songs for them
                // even though that's what we say we will play anyway
                if (rating.isSet())
                {
                        short value  = rating.get();
                        if (value > Constants.MIN_RATING_FOR_RATED_SOURCE && value >= maxValue)
                        {
                                maxValue  = value;
                                maxSource = ratingSource;
                        }
                }
                // lies, lies, lies.
                if (maxValue < Constants.MIN_RATING_FOR_RATED_SOURCE)
                {
                        maxSource = SOURCE_RANDOM;
                }
                return maxSource;
        }
        public void calculateDJs(ItemsProfile items, AlbumArtistData albumAndArtist)
        {
                // put in the default
                djs.set(djsAverage.get());
                djSource = SOURCE_DJS_SONG;
                if (djsAverage.count() <= 0)
                {
                        djSource = SOURCE_RANDOM;
                        Item albumItem = albumAndArtist.getAlbum(items, this);
                        Item artistItem = albumAndArtist.getArtist(items, this);
                        // don't calculate implicit ratings based on various artists
                        if (artistItem != null && ArtistInfo.isVariousArtists(artistItem.itemID))
                        {
                                artistItem = null;
                        }
```

```
                        if (albumItem != null && albumItem.djsAverage.count() > 0)
                        {
                                djs.set(albumItem.djsAverage.get());
                                djSource = SOURCE_DJS_ALBUM;
                        }
                        else if (artistItem != null && artistItem.djsAverage.count() > 0)
                        {
                                djs.set(artistItem.djsAverage.get());
                                djSource = SOURCE_DJS_ARTIST;
                        }
                }
        }
        public byte calculateImplicit(ItemsProfile items, AlbumArtistData albumAndArtist)
        {
                if (!rating.isSet())
                {
                        Item albumItem = albumAndArtist.getAlbum(items, this);
                        Item artistItem = albumAndArtist.getArtist(items, this);
                        // don't calculate implicit ratings based on various artists
                        if (artistItem != null && ArtistInfo.isVariousArtists(artistItem.itemID))
                        {
                                artistItem = null;
                        }
                        if (albumItem != null && albumItem.userRating.isSet())
                        {
                                short albumRating = albumItem.userRating.get();
                                if (albumRating == 0)
                                        return EXCLUDE_ME;
                                else
                                {
                                        rating.set(albumRating,
SongRating.RATING_SOURCE_FROM_ALBUM);
                                        return MAKE_ME_IMPLICIT;
                                }
                        }
                        else if (artistItem != null && artistItem.userRating.isSet())
                        {
                                short artistRating = artistItem.userRating.get();
                                if (artistRating == 0)
                                        return EXCLUDE_ME;
                                else
                                {
                                        rating.set(artistRating,
SongRating.RATING_SOURCE_FROM_ARTIST);
                                        return MAKE_ME_IMPLICIT;
                                }
                        }
                        else if (artistItem != null && artistItem.songAverage.count() > 0)
                        {
                                rating.set((short) artistItem.songAverageScore(info.album.artist),
SongRating.RATING_SOURCE_AVERAGE_SONG_RATING_BY_ARTIST);
                                return MAKE_ME_IMPLICIT;
                        }
                }
                return DO_NOTHING;
        }
        public void setBDS(short score)
        {
                bds = score;
        }
        public double getBDS()
        {
```

```
185                 return bds;
            }
            public void score(WeightMatrix w, StationList stations)
            {
                    // score bds
190                 bds = info.bdsScore(stations);
                    byte s = rating.getSource();
                    /*
                    // we're not using confidence right now. Take it out for speed
                    confidence = 0;
195                 if (ratingSource != SongRating.RATING_SOURCE_EXPLICIT)
                    {
                            if (djs != DEFAULT_DJS_SCORE)
                                    confidence += 10;
                            if (netp > 0)
200                                 confidence += 10;
                            if (info.commRating > 0)
                                    confidence += 10;
                    }
                    */
205                 // implicit rating is based on ratings data
                    ratingF    = (rating.get()    * w.matrix[s][WeightMatrix.RATING    ]);
                    djsF       = (djs.get()       * w.matrix[s][WeightMatrix.DJS       ]);
                    netpF      = (netp            * w.matrix[s][WeightMatrix.NETP      ]);
                    commRatingF = (info.commRating * w.matrix[s][WeightMatrix.COMM_RATING]);
210                 lastPlayedF = (lastPlayed      * w.matrix[s][WeightMatrix.LAST_PLAYED]);
                    bdsF       = (bds             * w.matrix[s][WeightMatrix.BDS       ]);
                    implicit   = ratingF + djsF + netpF + commRatingF;
                    // score is based on other factors
                    score      = implicit + lastPlayedF + bdsF;
215 //                confidence += w.matrix[s][WeightMatrix.CONFIDENCE];
            }
            public void setInfo(SongInfo stuff)
            {
                    info = stuff;
220         }
            public SongInfo getInfo()
            {
                    return info;
            }
225         public boolean isInfoSet()
            {
                    return (info != null);
            }
            public int getArtistID()
230         {
                    return info.album.artist.ID;
            }
            public int getAlbumID()
            {
235                 return info.album.ID;
            }
            public String getArtistName()
            {
                    return info.album.artist.title;
240         }
            public String getAlbumName()
            {
                    return info.album.title;
            }
245         public int getMediaID(short mediaType)
            {
```

```
        return info.media.getID(mediaType);
    }
    public String getSongName()
    {
        return info.title;
    }
    public String sourceString(byte source)
    {
        switch (source) {
        case SOURCE_RECENTLY_PLAYED:
            return "recent";
        case SOURCE_RATED:
            return "rated";
        case SOURCE_IMPLICIT_ALBUM:
            return "album";
        case SOURCE_IMPLICIT_ARTIST:
            return "artist";
        case SOURCE_IMPLICIT_SONG:
            return "s avg";
        case SOURCE_DJS:
            return "djs";
        case SOURCE_DJS_ALBUM:
            return "djAlb";
        case SOURCE_DJS_ARTIST:
            return "djArt";
        case SOURCE_BDS:
            return "bds";
        case SOURCE_POPULAR:
            return "pop";
        case SOURCE_RANDOM:
            return "random";
        case SOURCE_NETP:
            return "netp";
        case SOURCE_GENRES:
            return "genres";
        case SOURCE_ALL:
            return "all";
        default:
            return "?";
        }
    }
    public static String originText(byte origin, String singularDJ, String posessiveDJ)
    {
        switch (origin)
        {
            case SOURCE_RATED:
                return (singularDJ + " rated this song");
            case SOURCE_IMPLICIT_ALBUM:
                return (singularDJ + " rated this album");
            case SOURCE_IMPLICIT_ARTIST:
                return (singularDJ + " rated this artist");
            case SOURCE_IMPLICIT_SONG:
                return (singularDJ + " rated other songs by this artist");
            case SOURCE_DJS:
                return (posessiveDJ + " DJs rated this song");
            case SOURCE_DJS_ALBUM:
                return (posessiveDJ + " DJs rated this album");
            case SOURCE_DJS_ARTIST:
                return (posessiveDJ + " DJs rated this artist");
            case SOURCE_BDS:
                return (posessiveDJ + " radio stations play this song");
            case SOURCE_POPULAR:
```

```
                return "This song is popular on LAUNCHcast stations";
            case SOURCE_RANDOM:
                return "This song is a random pick";
            case SOURCE_NETP:
                return "Song recommendations";
            case SOURCE_FORCED_POPULAR:
                return "Popular - choose more genres for your music.";
        }
        return "";
    }
    public String toString()
    {
        return "songID:" + songID + ", "
                + "score:" + score + ", "
                + "implicit:" + implicit + ", "
              + "confidence: " + confidence + ", "
                + "lastPlayed:" + lastPlayed + ", "
                + "rating:" + rating + ", "
                + "ratingSource:" + rating.getSource() + ", "
                + "bds:" + bds + ", "
                + "djs:" + djs.get() + ", "
                + "source:" + sourceString(querySource) + Util.newLine;
    }
    public PlaylistEntry toPlaylistEntry(short mediaType)
    {
        PlaylistEntry result = new PlaylistEntry();
        result.albumID     = getAlbumID();
        result.artistID    = getArtistID();
        result.albumTitle  = info.album.title;
        result.artistTitle = info.album.artist.title;
        result.filepath    = info.media.getFilepath(mediaType);
        result.mediaID     = getMediaID(mediaType);
        result.songID      = songID;
        result.songTitle   = info.title;
        result.title       = info.title;
        return result;
    }

    public SimpleClip toSimpleClip(short mediaType)
    {
        return new SimpleClip(songID, getMediaID(mediaType), origin());
    }

    public String toDisplayString(int displayType, int count)
    {
        String delim  = "";
        String prefix = "";
        String suffix = "";
        String bgcolor = "";
        if (displayType == Util.DISPLAY_HTML)
        {
            if (count % 2 == 0)
                bgcolor = "#CCCCFF";
            else
                bgcolor = "white";
            delim = "</FONT></TD><TD BGCOLOR=" + bgcolor + "><FONT SIZE=\"-2\">";
            prefix = "<TR><TD BGCOLOR=" + bgcolor + "><FONT SIZE=\"-2\">";
            suffix = "</FONT></TD></TR>";
        }
        else {
            delim = "\t";
        }
```

```
                return (prefix + count
                        + delim + songID
                        + delim + sourceString(querySource)
                        + delim + sourceString(origin())
                        + delim + status.toDisplayString(displayType)
                        + delim + status.order
                        + delim + Util.fix(score, 2, 0)
                        + delim + Math.round(lastPlayed) + "/" + Math.round(lastPlayedF)
                        + delim + Math.round(bds)      + "/" + Math.round(bdsF)
                        + delim + Math.round(implicit)
                        + delim + Util.fix(rating.get(), 0, 2) + "/" + Util.fix(ratingF, 0, 2) + " (" +
rating.getSource() + ")"
                        + delim + Math.round(djs.get()) + "/" + Math.round(djsF)
                        + delim + Math.round(netp)      + "/" + Math.round(netpF)
                        + delim + Math.round(info.commRating)  + "/" + Math.round(commRatingF)
                        + delim + getAlbumID()
                        + delim + getArtistID()
                        + delim + getArtistName()
                        + delim + getSongName()
                        + delim + getAlbumName()
                        + delim + info.album.genresString()
                        + suffix
                );
        }
        public String originTclList()
        {
                return "{" + songID + " " + origin() + " " + Math.round(implicit) + "} ";
        }
        public static String[] namesArray()
        {
                String[] names = { "#",
                                                "songID",
                                                "query",
                                                "origin",
                                                "status",
                                                "ord",
                                                "score",
                                                "lastP.",
                                                "bds",
                                                "impl.",
                                                "rating(t)",
                                                "djs",
                                                "netP.",
                                                "comm",
                                                "albumID",
                                                "artisID",
                                                "artist",
                                                "title",
                                                "album",
                };
                return names;
        }
}
```

SongGroup

```
package com.launch.PlaylistGenerator;
import java.util.Vector;
public class SongGroup extends Vector
{

        public SongData pickRandom(int factor)
        {
                int leftInList = size();
                if (leftInList <= 0)
                        return null;
                double rand         = Util.random(leftInList - 1) + 0.00001;
                int pickIndex       = (int) Math.round((Math.pow(rand, factor) / Math.pow(leftInList - 1, factor))
* (leftInList - 1));
                SongData pick       = (SongData) elementAt(pickIndex);
                double pickDouble   = pickIndex;
                pick.status.percentile = (short) Math.round((pickDouble / size()) * 100);
                removeElementAt(pickIndex);
                return pick;
        }

}
```

SongGroup.java Page 1 of 1 11/05/99 1:28 PM

SongInfo

```
package com.launch.PlaylistGenerator;
import java.util.Vector;
public class SongInfo
{
        int songID;
        byte commRating = Constants.DEFAULT_COMMRATING;
        private boolean explicit = false;

        AlbumInfo album;
        String title;
        private Vector bdsRanks;
        public MediaList media;

        public SongInfo(int songID)
        {
                this.songID = songID;
                media = new MediaList();
        }

        public void addBDSRank(BDSRank rank)
        {
                if (bdsRanks == null)
                        bdsRanks = new Vector(1, 1);

                bdsRanks.addElement(rank);
        }

        public int getArtistID() /* throws Exception */
        {

                return album.artist.ID;

                /*
                if (album == null)
                {
                        throw new Exception("album is not set for SongInfo songID " + songID + "(" + title +
")");
                }

                return album.getArtistID();
                */
        }

        public int getAlbumID() /* throws Exception */
        {
                /*
                if (album == null)
                {
                        throw new Exception("album is not set for SongInfo songID " + songID + "(" + title +
")");
                }
                */

                return album.ID;
        }


        public double bdsScore(StationList stations)
        {
```

```
        if (bdsRanks == null || stations.size() <= 0)
                return Constants.DEFAULT_BDS_SCORE;

        int i           = 0;
        int pointBar    = Constants.BDS_SCORE_POINTBAR;
        float maxPoints  = Constants.BDS_SCORE_MAX_POINTS;
        float totalpoints = 0;
        float numStations = 0;

        BDSRank rank;
        Station sta;

        for (int j = 0; j < bdsRanks.size(); j++)
        {
                rank = (BDSRank) bdsRanks.elementAt(j);
                sta  = stations.get(rank.stationID);

                if (sta != null)
                {
                        totalpoints += (maxPoints - rank.rank);
                        numStations++;
                }
        }

        double potentialStations = stations.size();

        double score = ((((totalpoints / potentialStations) / maxPoints) + (numStations / potentialStations)
) * 150.0);

        return score;
    }

    public String bdsString()
    {
        String result = "";

        if (bdsRanks == null)
                return "(none)";

        for (int i = 0; i < bdsRanks.size(); i++)
        {
                result = result.concat(bdsRanks.elementAt(i).toString() + ",");
        }
        return "(" + result + ")";
    }

    public String toString()
    {
        return "songID=" + songID + ", "
                + "title=" + title + ", "
                + "commRating=" + commRating + ", "
                + "media=" + media.toString()
                + "bdsRanks=" + bdsString()
                + "album=" + album.toString();
    }

    public void setExplicitLyrics(boolean badStuff)
    {
        explicit = badStuff;
    }

    public boolean hasExplicitLyrics()
```

```
        {
            return explicit;
125     }

}
```

SongInfoCache

```
package com.launch.PlaylistGenerator;
import java.util.Hashtable;
import java.util.Enumeration;
import javax.servlet.ServletOutputStream;
import java.util.Date;
import java.util.Vector;
public class SongInfoCache
{
        private Hashtable songs;
        private Hashtable albums;
        private Hashtable artists;
        private SongInfo songList[];
        private Hashtable ads;
        private Hashtable news;
        private Hashtable tips;
        private Clip adList[];
        private Clip newsList[];
        private Clip tipList[];
        private IntHash mediaTypes;
        public  PopularSongs popular;
        public  RatingsCache ratingsCache;
        private GenreIndex genres;
        public final static byte TYPE_SONG  = 1;
        public final static byte TYPE_ALBUM  = 2;
        public final static byte TYPE_ARTIST = 3;
        public final static byte TYPE_AD    = 4;
        public final static byte TYPE_NEWS  = 5;
        public final static byte TYPE_TIP   = 6;
        private ServletOutputStream out;
        public Date lastUpdate;
        public SongInfoCache(ServletOutputStream out)
        {
                // use memory most efficiently with load factor 1
                songs     = new Hashtable(50000);
                albums    = new Hashtable(3000);
                artists   = new Hashtable(1500);
                ads       = new Hashtable();
                news      = new Hashtable();
                tips      = new Hashtable();
                mediaTypes = new IntHash();
                genres    = new GenreIndex(100, 1);
                populate();
                lastUpdate = new Date();
        }
        public SongList getPopular(short mediaType)
        {
                return popular.get(mediaType);
        }
        public SongList getInGenres(GenreList myGenres)
        {
                return genres.getInGenreList(myGenres);
        }
        public SongList getInGenre(int genreID)
        {
                return genres.getInGenre(genreID);
        }
        public int countInGenres(GenreList myGenres)
        {
                return genres.countInGenreList(myGenres);
```

```
        }
        private void populate()
        {
                try
                {
                        DBConnection conn = new DBConnection();
                        DBResultSet rs   = conn.executeSQL("exec sp_lcoGetSongDataCache_xsxx");
                        int songID, mediaType, rank, stationID, rowCount;
                        short genreID;
                        String filePath;
                        SongInfo aSong;
                        ArtistInfo anArtist;
                        AlbumInfo anAlbum;
                        rowCount = 0;
                        while (!rs.getBOF() && !rs.getEOF())
                        {
                                songID    = rs.getInt("songID");
                                mediaType  = rs.getInt("mediaType");
                                aSong = (SongInfo) init(songID, SongInfoCache.TYPE_SONG);
                                filePath = rs.getString("server") + rs.getString("directory") + "\\" +
rs.getString("filePath");
                                aSong.media.add((short) mediaType, rs.getInt("mediaID"), filePath);
                                aSong.title = rs.getString("song");
                                anArtist = (ArtistInfo) init(rs.getInt("artistID"),
SongInfoCache.TYPE_ARTIST);
                                anArtist.title = rs.getString("artist");
                                anArtist.songs.put(new Integer(songID), aSong);
                                anAlbum = (AlbumInfo) init(rs.getInt("albumID"),
SongInfoCache.TYPE_ALBUM);
                                anAlbum.title = rs.getString("album");
                                aSong.setExplicitLyrics(rs.getInt("explicit") == 1);
                                // add year and date added
                                anAlbum.artist = anArtist;
                                aSong.album = anAlbum;
                                mediaTypes.increment(mediaType);
                                rowCount++;
                                rs.next();
                        }
                        Util.debug("SongInfoCache:populate loaded " + rowCount + " media");
                        rs = conn.executeSQL("exec sp_lcoGetCommRatingCache_xsxx");
                        rowCount = 0;
                        while (!rs.getBOF() && !rs.getEOF())
                        {
                                songID = rs.getInt("songID");
                                aSong = (SongInfo) get(songID, SongInfoCache.TYPE_SONG);

                                if (aSong != null)
                                {
                                        aSong.commRating = (byte) rs.getInt("commRating");
                                        rowCount++;
                                }
                                rs.next();
                        }
                        Util.debug("SongInfoCache:populate loaded " + rowCount + " commRatings");
                        rs = conn.executeSQL("exec sp_lcoGetGenreCache_xsxx");
                        while (!rs.getBOF() && !rs.getEOF())
                        {
                                genreID = (short) rs.getInt("genreID");
                                songID = rs.getInt("songID");
                                aSong = (SongInfo) get(songID, SongInfoCache.TYPE_SONG);
                                if (aSong != null && aSong.album != null)
                                {
```

```
                                aSong.album.addGenre(genreID);
                                genres.add(genreID, aSong);
                                rowCount++;
                        }
                        rs.next();
                }
                Util.debug("SongInfoCache:populate loaded " + rowCount + " genre mappings");
                rowCount = 0;
                rs = conn.executeSQL("exec sp_IcoGetBDSCache_xsxx");
                while (!rs.getBOF() && !rs.getEOF())
                {
                        songID = rs.getInt("songID");
                        aSong = (SongInfo) get(songID, TYPE_SONG);
                        if (aSong != null)
                        {
                                rank     = rs.getInt("rank");
                                stationID = rs.getInt("stationID");
                                rowCount++;
                                aSong.addBDSRank(new BDSRank((short) stationID, (byte) rank));
                        }
                        rs.next();
                }
                Util.debug("SongInfoCache:populate loaded " + rowCount + " bds Ranks");
                // import ads
                rowCount = 0;
                rs = conn.executeSQL("exec sp_IcoGetAdCache_xsxx");
                Clip ad;
                int clipID;

                while (!rs.getBOF() && !rs.getEOF())
                {
                        clipID = rs.getInt("clipID");
//              filePath = rs.getString("server") + rs.getString("directory") + "/" +
rs.getString("filePath");

                        ad    = (Clip) init(clipID, TYPE_AD);
//                      ad.name    = rs.getString("clipName");
                        ad.media.add((short) rs.getInt("mediaType"), rs.getInt("mediaID"), null);
                        rowCount++;
                        rs.next();
                }
                Util.debug("SongInfoCache:populate loaded " + rowCount + " ad media");
                // import news
                rs = conn.executeSQL("exec sp_IcoGetNewsCache_xsxx");
                rowCount = 0;
                Clip newsbit;
                while (!rs.getBOF() && !rs.getEOF())
                {
                        clipID = rs.getInt("clipID");
                        filePath = rs.getString("server") + rs.getString("directory") + "\\" +
rs.getString("filePath");

                        newsbit    = (Clip) init(clipID, TYPE_NEWS);
                        newsbit.name    = rs.getString("clipName");
                        newsbit.media.add((short) rs.getInt("mediaType"), rs.getInt("mediaID"),
filePath);

                        rowCount++;
                        rs.next();
                }
                Util.debug("SongInfoCache:populate loaded " + rowCount + " news media");
                // import tips
                rs = conn.executeSQL("exec sp_IcoGetTipCache_xsxx");
                rowCount = 0;
                Clip tip;
```

```
                        while (!rs.getBOF() && !rs.getEOF())
                        {
                                clipID = rs.getInt("clipID");
                                filePath = rs.getString("server") + rs.getString("directory") + "\\" +
rs.getString("filePath");
                                tip    = (Clip) init(clipID, TYPE_TIP);
                                tip.name   = rs.getString("clipName");
                                tip.media.add((short) rs.getInt("mediaType"), rs.getInt("mediaID"), filePath);
                                rowCount++;
                                rs.next();
                        }
                        Util.debug("SongInfoCache:populate loaded " + rowCount + " tip media");
                        conn.close();
                }
                catch (DBException oops)
                {
                        System.out.println("DBException in cache populate: " + oops.getMessage());
                }
                // populate the songs array
                songList = new SongInfo[songs.size()];
                int i = 0;
                for (Enumeration e = songs.keys(); e.hasMoreElements() ;) {
                        songList[i] = (SongInfo) songs.get((Integer) e.nextElement());
                        i++;
                }
                // populate the ads array
                adList = new Clip[ads.size()];
                i = 0;
                for (Enumeration e = ads.keys(); e.hasMoreElements() ;) {
                        adList[i] = (Clip) ads.get((Integer) e.nextElement());
                        i++;
                }
                // populate the news array
                newsList = new Clip[news.size()];
                i = 0;
                for (Enumeration e = news.keys(); e.hasMoreElements() ;) {
                        newsList[i] = (Clip) news.get((Integer) e.nextElement());
                        i++;
                }
                // populate the tips array
                tipList = new Clip[tips.size()];
                i = 0;
                for (Enumeration e = tips.keys(); e.hasMoreElements() ;) {
                        tipList[i] = (Clip) tips.get((Integer) e.nextElement());
                        i++;
                }
                // make popular lists
                popular = new PopularSongs(songs, mediaTypes);
                Util.debug("SongInfoCache:populate done");
        }
        private Hashtable getHash(byte type)
        {
                if (type == TYPE_SONG)
                        return songs;
                else if (type == TYPE_ALBUM)
                        return albums;
                else if (type == TYPE_ARTIST)
                        return artists;
                else if (type == TYPE_AD)
                        return ads;
                else if (type == TYPE_NEWS)
                        return news;
```

```
            else if (type == TYPE_TIP)
                return tips;
            return null;
    }
    public Object init(int ID, byte type)
    {
            if (getHash(type).containsKey(new Integer(ID)))
            {
                return get(ID, type);
            }
            else {
                return put(ID, type);
            }
    }
    public Object get(Integer ID, byte type)
    {
            return (getHash(type)).get(ID);
    }
    public Object get(int ID, byte type)
    {
            return get(new Integer(ID), type);
    }
    private Object makeNew(int ID, byte type)
    {
            if (type == TYPE_SONG)
                return new SongInfo(ID);
            else if (type == TYPE_ALBUM)
                return new AlbumInfo(ID);
            else if (type == TYPE_ARTIST)
                return new ArtistInfo(ID);
            else if (type == TYPE_AD)
                return new Clip(ID, Clip.TYPE_AD);
            else if (type == TYPE_NEWS)
                return new Clip(ID, Clip.TYPE_NEWS);
            else if (type == TYPE_TIP)
                return new Clip(ID, Clip.TYPE_TIP);
            return null;
    }
    private Object put(int ID, byte type)
    {
            Hashtable hash = getHash(type);
            Object thing = makeNew(ID, type);
            hash.put(new Integer(ID), thing);
            return thing;
    }
    public SongInfo randomSong()
    {
            long index = Util.random(songList.length - 1);
            if (index > songList.length - 1)
                return null;
            return songList[(int) index];
    }
    public Enumeration keys(byte type)
    {
            if (type == TYPE_SONG)
                return songs.keys();
            else if (type == TYPE_ALBUM)
                return albums.keys();
            else if (type == TYPE_ARTIST)
                return artists.keys();
            else if (type == TYPE_AD)
                return ads.keys();
```

```
                else if (type == TYPE_NEWS)
                        return news.keys();
                else if (type == TYPE_TIP)
                        return tips.keys();
                return null;
        }
        public int size(byte type)
        {
                Hashtable hash = getHash(type);
                if (hash != null)
                        return hash.size();
                return 0;
        }
        private Clip[] getClipList(byte type)
        {
                if (type == TYPE_AD)
                        return adList;
                else if (type == TYPE_NEWS)
                        return newsList;
                else if (type == TYPE_TIP)
                        return tipList;
                return null;
        }
        public Clip randomClip(byte type)
        {
                Clip[] clips = getClipList(type);
                if (clips == null || clips.length <= 0)
                        return null;
                return clips[(int) Util.random(clips.length - 1)];
        }
        public Vector randomClipList(byte type, short mediaType, int max)
        {
                Vector list = new Vector();
                Clip bip;
                // stop if we have enough or we've iterated too many times
                for (int i = 0; i < (max * 10) && list.size() < max; i++)
                {
                        int iterations = max;
                        boolean cool = false;
                        boolean done = false;
                        do
                        {
                                bip = randomClip(type);
                                iterations--;
                                // maybe we didn't get one
                                if (bip == null)
                                {
                                        done = true;
                                }
                                else
                                {
                                        // we got one that fits!
                                        cool = (bip.media.inType(mediaType) && !list.contains(bip));
                                        // we've got to stop sometime
                                        done = (cool || iterations < 0);
                                }
                        }
                        while (!done);
                        // if it was cool, go ahead
                        if (cool)
                                list.addElement(bip);
                }
```

```
            return list;
        }
}
```

SongInfoCache.java Page 9 of 9 11/05/99 1:32 PM

SongInfoCacheUpdater

```
package com.launch.PlaylistGenerator;
import javax.servlet.http.HttpServlet;
import java.util.Date;
public class SongInfoCacheUpdater extends Thread
{
        PlaylistGeneratorServlet servlet;

        public SongInfoCacheUpdater(PlaylistGeneratorServlet servlet)
        {
                this.servlet = servlet;
        }
        public void run()
        {
                Thread.currentThread().setName("SongInfoCacheUpdater");

                // update every day
                long timeToSleep = Util.MILLISECONDS_IN_SECOND *
                                                Util.SECONDS_IN_MINUTE    *
                                                Util.MINUTES_IN_HOUR      *
                                                Util.HOURS_IN_DAY;

                while (true)
                {

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

                        try
                        {

                                Util.debug("updating song cache at " + new Date());
                                Util.debug("last update was at " + servlet.songCache.lastUpdate);

                                // make a new cache
                                SongInfoCache cache = new SongInfoCache(null);

                                // make sure to copy over the ratingsCache too!!!
                                cache.ratingsCache = servlet.songCache.ratingsCache;

                                // install the new cache
                                servlet.songCache = cache;
                                Util.debug("finished updating song cache at " + new Date());
                                Util.debug("last update is now at " + servlet.songCache.lastUpdate);
                        }
                        catch (Throwable e)
                        {
                                System.err.println("SongInfoCacheUpdater caught an exception: " +
e.toString());
                                e.printStackTrace();
                        }

                }
        }
}
```

SongList

```
package com.launch.PlaylistGenerator;
import java.util.Vector;
import java.util.Hashtable;
import java.util.Enumeration;
public class SongList implements Cloneable
{
        private Vector     list = new Vector();
        private Hashtable unique = new Hashtable();
        private boolean ordered  = false;
        public SongList()
        {
        }
        /**
         * Creates a SongList from a Hashtable of songs
         */
        public SongList(Hashtable songs)
        {
                SongInfo info = null;
                Integer songID;
                for (Enumeration e = songs.keys(); e.hasMoreElements();)
                {
                        songID = (Integer) e.nextElement();
                        info = (SongInfo) songs.get(songID);
                        addElement(info);
                }
        }
        public SongList(Hashtable songs, short mediaType)
        {
                Integer songID;
                SongInfo info = null;
                for (Enumeration e = songs.keys(); e.hasMoreElements();)
                {
                        songID = (Integer) e.nextElement();
                        info = (SongInfo) songs.get(songID);
                        if (info.media.inType(mediaType))
                        {
                                addElement(info);
                        }
                }
        }
        public void addElement(SongInfo info)
        {
                Integer ID = new Integer(info.songID);
                // check unique constraint
                if (unique.get(ID) == null)
                {
                        list.addElement(info);
                        unique.put(ID, info);
                }
        }
        public void addElements(SongList list)
        {
                if (list == null)
                        return;
                for (int i = 0; i < list.size(); i++)
                {
                        addElement(list.elementAt(i));
                }
        }
```

```
public void sort()
{
        sort(this, 0, list.size() - 1);
        ordered = true;
}
public int size()
{
        return list.size();
}
public SongInfo elementAt(int index)
{
        return (SongInfo) list.elementAt(index);
}
public void setSize(int newSize)
{
        list.setSize(newSize);
}
private void sort(SongList a, int from, int to)
{

        // quicksort
        // If there is nothing to sort, return
        if ((a == null) || (a.size() < 2)) return;
        int i = from, j = to;
        SongInfo center = a.elementAt((from + to) / 2);

        do {
                while((i < to) && (center.commRating < a.elementAt(i).commRating)) i++;
                while((j > from) && (center.commRating > a.elementAt(j).commRating)) j--;
                if (i < j) {
                        SongInfo temp = a.elementAt(i);
                        a.setElementAt(a.elementAt(j), i);
                        a.setElementAt(temp, j);   // swap elements
                }
                if (i <= j) { i++; j--; }
        } while(i <= j);
        if (from < j) sort(a, from, j); // recursively sort the rest
        if (i < to) sort(a, i, to);
}
public void setElementAt(SongInfo info, int index)
{
        list.setElementAt(info, index);
}
public SongInfo pickRandom()
{
        if (size() <= 0)
                return null;
        int lucky = (int) Util.random(size() - 1);
        if (lucky < 0)
                return null;
        SongInfo info = elementAt(lucky);
        list.removeElementAt(lucky);
        return info;
}
public Object clone()
{
        SongList result = new SongList();
        result.ordered  = this.ordered;
        result.unique   = (Hashtable) unique.clone();
        result.list     = (Vector) list.clone();
        return result;
}
```

```
}
```
SongList.java Page 3 of 3 11/05/99 1:34 PM

SongRating

```
package com.launch.PlaylistGenerator;
public class SongRating
{

        public final static byte RATING_SOURCE_NONE      = 0;
        public final static byte RATING_SOURCE_EXPLICIT  = 1;
        public final static byte RATING_SOURCE_FROM_ALBUM = 2;
        public final static byte RATING_SOURCE_FROM_ARTIST = 3;
        public final static byte RATING_SOURCE_AVERAGE_SONG_RATING_BY_ARTIST = 4;

        private short rating = (short) Constants.DEFAULT_RATING;
        private boolean set = false;
        private byte type;

        public boolean isSet()
        {
                return set;
        }

        public short set(short newRating, byte newType)
        {

                rating = newRating;
                type = newType;
                set = true;

                return rating;
        }

        public short get()
        {
                return rating;
        }

        public byte getSource()
        {
                return type;
        }
}
```

SongRating.java Page 1 of 1 11/05/99 1:38 PM

Station

```
package com.launch.PlaylistGenerator;
public class Station
{
        int ID;

        public Station(int stationID)
        {
                ID = stationID;
        }
}
```

Station.java Page 1 of 1 11/05/99 1:26 PM

StationList

```
package com.launch.PlaylistGenerator;
import java.util.Vector;
public class StationList
{

        private Vector slist;

        public StationList()
        {
                slist = new Vector();
        }

        public Station stationAt(int i)
        {
                return (Station) slist.elementAt(i);
        }

        public void addElement(Station s)
        {
                slist.addElement(s);
        }

        public int size()
        {
                return slist.size();
        }

        public String inList()
        {
                Integer list[] = new Integer[size()];

                int last = 0;

                for (int i = 0; i < slist.size(); i++)
                {
                        list[i] = new Integer(stationAt(i).ID);
                }

                return Util.join(", ", list);
        }
        public Station get(int stationID)
        {
                for (int i = 0; i < slist.size(); i++)
                {
                        if (stationAt(i).ID == stationID)
                        {
                                return stationAt(i);
                        }
                }
                return null;
        }
}
```

StationList.java Page 1 of 1 11/05/99 1:26 PM

Util

```
package com.launch.PlaylistGenerator;
import java.io.OutputStream;
import java.util.Date;
import javax.servlet.ServletOutputStream;
import java.io.IOException;
public class Util
{
	public static final int MILLISECONDS_IN_SECOND = 1000;
	public static final int SECONDS_IN_MINUTE = 60;
	public static final int MINUTES_IN_HOUR = 60;
	public static final int HOURS_IN_DAY = 24;
	public static final int DAYS_IN_WEEK = 7;
	public static final int DAYS_IN_MONTH = 30;
	public static final int DISPLAY_TEXT = 0;
	public static final int DISPLAY_HTML = 1;
	public static final String newLine = "\r\n";
	public static final short average(double count, double sum)
	{
		if (count == 0)
			return 0;
		return (short) Math.round(sum / count);
	}
	public static final long random(int ceiling)
	{
		return Math.round(Math.random() * ceiling);
	}
	public static final String join (String delim, Object values[])
	{
		String result = "";
		int i = 0;
		for (; i < values.length; i++)
			result = result.concat(values[i].toString() + delim);
		if (i > 0)
			result = result.substring(0, (result.length() - delim.length()));
		return result;
	}
	public static final String fix(double number, int precision, int zeroFill)
	{
		double power = Math.pow(10, precision);
		double fixed = Math.round(number * power) / power;
		String mantissa = new Long(Math.round(fixed)).toString();
		String result = mantissa;
		for (int i = mantissa.length(); i < zeroFill; i++)
			result = new String("0" + result);
		return result;
	}
	public static final void out(ServletOutputStream stream, String whatever)
	{
		try
		{
			if (stream == null)
				System.out.println(whatever);
			else
				stream.println(whatever);
		}
		catch (IOException e)
		{
		}
	}
```

```
        public static final void debug(String info)
        {
                System.out.println(info);
        }
        public final static String tab(int times)
        {
                String result = "";
                for (int i = 0; i < times; i++)
                {
                        result = result.concat("    ");
                }
                return result;
        }
        public static final void markQueryFinished(String threadName, Date startDate)
        {
                Util.debug(newLine + threadName + " started getting data after "
                                + ((new Date().getTime() - startDate.getTime()) / 1000.0)
                                + " seconds" + newLine);
        }
        public static final void printElapsedTime(String threadName, Date startDate)
        {
                Util.debug(newLine + new Date().toString() + " " + threadName + " took "
                                + ((new Date().getTime() - startDate.getTime()) / 1000.0)
                                + " seconds" + newLine);
        }
        public static final String tab()
        {
                return tab(1);
        }
}
```

Util.java Page 3 of 3 11/05/99 1:37 PM

WeightMatrix

```
package com.launch.PlaylistGenerator;
public class WeightMatrix
{
        public final static byte RATING     = 0;
        public final static byte DJS        = 1;
        public final static byte NETP       = 2;
        public final static byte COMM_RATING = 3;
        public final static byte LAST_PLAYED = 4;
        public final static byte BDS        = 5;
        public final static byte CONFIDENCE  = 6;
        // rating, djs, netp, commRating, lastPlayed, bds, conf
        public double matrix[][] = {
                                                {0.00, 0.33, 0.00, 0.10, 0.25, 0.20, 0.0},   // no rating
                                                {0.70, 0.00, 0.00, 0.00, 0.30, 0.00, 100.0}, // explicit rating
                                                {0.45, 0.05, 0.00, 0.05, 0.20, 0.20, 50.0},  // album rating only
                                                {0.40, 0.10, 0.00, 0.05, 0.20, 0.20, 30.0},  // artist only
                                                {0.35, 0.15, 0.00, 0.05, 0.20, 0.20, 20.0}   // cross-propagated
song ratings
                                                };
}
WeightMatrix.java       Page 1 of 1       11/05/99 1:32 PM
```

CLAIMS

What is claimed is:

1. A method for broadcasting data streams through a computer network to a user's computer, the steps comprising:
providing a database of data streams;
selecting a data stream according to a selection method;
transmitting one of said data streams to the user's computer;
receiving feedback expressing a preference from the user regarding said transmitted data stream; and
updating said selection method to better reflect said preference of the user; whereby
data streams transmitted to the user are biased according to said preference.

2. The method for broadcasting data streams through a computer network to a user's computer of Claim 1, further comprising:
said selection method including generating a list of data streams to transmit to the user's computer;
transmitting one of said listed data streams to the user's computer; and
updating said list of data streams to better reflect said preference of the user; whereby
data streams transmitted to the user are biased according to said preference.

3. The method for broadcasting data streams through a computer network of Claim 1, the steps further comprising:
receiving feedback expressing preferences from sources other than the user.

4. The method for broadcasting data streams through a computer network of Claim 3, wherein the step of receiving preferences from sources other than the user further comprises:
receiving feedback expressing preferences from the group consisting of other users, commercial radio stations, and lists of popular songs.

5. The method for broadcasting data streams through a computer network of Claim 1, further comprising:
informing the user generally regarding said database and said data streams;
querying the user as to data stream preference prior to generating an initial transmission list of data streams; whereby
said initial list reflects general preferences of the user.

6. The method for broadcasting data streams through a computer network of Claim 1, wherein said data streams are selected from the group consisting of songs and videos.

7. The method for broadcasting data streams through a computer network of Claim 1, wherein said transmitted data stream is removed from said transmission list.

8. The method for broadcasting data streams through a computer network of Claim 7, wherein said data

stream removed from said transmission list is listed on a transmitted data stream list.

9. The method for broadcasting data streams through a computer network of Claim 1, wherein said step of transmitting one of said data streams further comprises transmitting said one of said data streams in conformance with applicable copyright law.

10. The method for broadcasting data streams through a computer network of Claim 9, wherein said conformance with applicable copyright law applies to all transmitted datastreams.

11. A data stream system for providing preferred data streams to a user, comprising:

a connection to a computer network, said computer network connected to a computer of the user;

a database of data streams, said database available to said computer network;

a data stream controller, said data stream controller transmitting data streams to said user's computer according to a selection program;

a user interface, said user interface coupled to said user's computer and receiving said data streams for the user and providing a feedback mechanism for the user so that the user may indicate a preference regarding data streams transmitted by said data stream controller;

said selection program receiving indications from the user, said selection program modifying its selection of data streams for transmission to said user's computer according to said user preference; whereby

data streams selected by said selection program are biased according to said user preference.

12. The data stream system for providing preferred data streams to a user of Claim 11, wherein said computer network comprises the Internet.

13. The data stream system for providing preferred data streams to a user of Claim 11, wherein said database is a song database and the data streams are songs.

14. The data stream system for providing preferred data streams to a user of Claim 11, wherein said database is a music video database and the data streams are music videos.

15. The data stream system for providing preferred data streams to a user of Claim 11, wherein said user interface comprises an electronic media player.

16. The data stream system for providing preferred data streams to a user of Claim 15, wherein said electronic media player is selected from the group consisting of RealPlayer, Apple QuickTime, and Windows Media Player.

17. The data stream system for providing preferred data streams to a user of Claim 11, wherein said selection program creates a list of data streams for transmission to the user.

18. The data stream system for providing preferred data streams to a user of Claim 17, wherein said selection program modifies said list of data streams for transmission to the user according to said user preference.

19. The data stream system for providing preferred data streams to a user as set forth in Claim 11, further comprising:

said data stream controller transmitting said data streams in compliance with applicable copyright law.

20. The data stream system for providing preferred data streams to a user as set forth in Claim 19, further comprising:

said data stream controller transmitting all data streams in compliance with applicable copyright law.

21. A user interface for an Internet datastream transmission system, comprising:

a media player, said playing data streams;

a rating tool, said rating tool indicating a rating for a data stream currently played by said media player; and

a data stream information display, said data stream information display displaying information for said data stream currently played by said media player; whereby

a user can indicate a preference regarding said data stream currently played by said media player.

22. A user interface for an Internet datastream transmission system as set forth in Claim 21, further comprising:

a playlist generator, said playlist generator generating playlists of data streams for said media player, said playlist generator selecting data streams according to preferences indicated by said user.

23. A user interface for an Internet datastream transmission system as set forth in Claim 22, further comprising:

said data streams selected by said playlist generator being in compliance with applicable copyright law.