(54) Title: BROWSER INTERPRETABLE DOCUMENT FOR CONTROLLING A PLURALITY OF MEDIA PLAYERS AND SYSTEMS AND METHODS RELATED THERETO

(57) Abstract: A browser interpretable document comprising a first media file or a pointer to the first media file; a graphical user interface definition, or a pointer to the graphical user interface definition, the graphical user interface definition comprising a first user interface element and a function associated with the first user interface element; and, a set of commands, or a pointer to the set of commands, for controlling the operation of a plurality of media players; wherein, when the browser interpretable document is rendered by a conventional web browser, the web browser can render a graphical user interface based on the graphical user interface definition, and wherein the rendered graphical user interface controls one of the plurality of media players to facilitate playing of the first media file.
BROWSER INTERPRETABLE DOCUMENT FOR CONTROLLING A PLURALITY OF MEDIA PLAYERS AND SYSTEMS AND METHODS RELATED THERETO

[0001] This application includes material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent disclosure, as it appears in the Patent and Trademark Office files or records, but otherwise reserves all copyright rights.

FIELD

[0002] The instant disclosure relates to the field of media player software, and more particularly describes a browser interpretable document for controlling a plurality of media players.

BACKGROUND

[0003] The advent of Motion Picture Entertainment Group Level 3 ("MP3") digital encoding for audio files has revolutionized the distribution of music by reducing the distribution costs to near zero. This, in turn, has encouraged artists to create new works and to make their works available in digital form. However, the MP3 media type has some disadvantages. For example, digital rights management capabilities are not built into the MP3 media type. As a result, some have sought to design new media types that facilitate digital rights management. Others have created new media types in an effort to improve the techniques used when converting analog sources, such as sound or light, into and/or from a digital format. Still others have created new media types that focus on reducing the amount of data that needs to be transmitted and/or stored to play the content with sufficient accuracy to please at least a majority of users, or for other, specific purposes. This has lead to an explosion of different media types in common use.

[0004] The rapid growth in digitally-available content has also spurred the creation of a variety of media players, including both dedicated, stand-alone devices such as, without limitation, the iPod line of personal music players distributed by Apple Computer, Inc. of Cupertino, California, the Zune distributed by Microsoft, Inc. of Redmond, Washington, and the Zen line of personal music players distributed by Creative Technology, Ltd. of
Singapore, and software-based media players for use on portable computers. To facilitate
the user's access to digital content, many media players can play content encoded using a
variety of media types. By way of example, the Windows Media Player software
distributed by Microsoft, Inc. of Redmond, Washington, can play content encoded using
a variety of media types, including Windows Media Audio ("WMA"), Windows Media
Video ("WMV"), Motion Picture Entertainment Group ("MPEG"), MP3, WAVE, and
Musical Instrument Digital Interface ("MIDI"). Windows Media Player also allows users
to add support for new and alternative media types by simply installing an appropriate
Coder/Decoder ("CODEC").

[0005] Although many media players can play content encoded in any of a variety of
media types, certain media types are proprietary, and the right to display content encoded
using that media type is frequently tightly controlled. By way of example, the encoding
technique employed by Apple Computer Inc.'s Quicklime, and RealNetworks' RealVideo and RealAudio media types are generally proprietary. As a result, users are
forced to use a proprietary media player, rather than their preferred media player, if they
wish to play content created using such a proprietary media type. This can be a problem
both for users attempting to access a variety of media files within their own personal
library, and for users attempting to access media files from other libraries. By way of
example, some users may share a playlist, or ordered list of content to be played, with
other users, and the other users may not realize that their preferred media player will not
be able to play all of the content in the playlist.

[0006] Some software-based media players permit users to create customized "skins", or
user interfaces, for that particular media player. Such skins allow users to more easily
access the commands and controls they most frequently use or need.

[0007] In addition, each media player typically has its advantages and disadvantages
when compared to the other media players available, and users tend to use the player with
the features and user interface that is most convenient and/or comfortable for them. As a
result, users tend to grow comfortable with a particular media player user interface,
especially a customized user interface, and when they are forced to switch from a media
player containing such a user interface to one that uses a different interface, such as
happens when content is encoded with a proprietary media type, they can become frustrated.

**SUMMARY**

[0008] What is needed is a means for providing a standardized interface through which a plurality of media players can be controlled. Accordingly, the instant disclosure is directed to a browser interpretable document for controlling a plurality of media players that substantially obviates one or more of the problems due to limitations and disadvantages of the related art. The media players described in the instant disclosure may be capable of playing audio, video, and/or other types of content; however, for clarity, the instant disclosure will focus on playing audio content encoded in a variety of media types, some of which may require a specific media player. Such a focus is adopted to clarify the browser interpretable document and its operation with the media players, and should not be read as limiting the browser interpretable document or the user interfaces created thereby to media players for audio content.

[0009] Many devices, including, without limitation, personal computers ("PC's"), laptops, personal digital assistants ("PDA's"), cellular telephones, gaming consoles, and portable media players ("PMP's") are now capable of playing content. These devices are also frequently capable of running at least one browser. Browsers are software applications that are capable of interpreting a document stored in Hypertext Markup Language ("HTML"), Extensible Markup Language ("XML") or other such markup language and presenting the document's contents to a user. Browsers also frequently support at least one scripting language, such as, without limitation Visual Basic Script ("VBScript"), distributed by the Microsoft Corporation of Redmond, Washington; and JavaScript™, distributed by Sun Microsystems. Such scripting languages can add a measure of programmability to the browser, rather than limiting the browser to simply displaying marked-up documents to users.

[0010] Some embodiments comprise a method of controlling a plurality of media players using a standardized interface, comprising defining a graphical user interface through which a plurality of media players are to be controlled, wherein the graphical user
interface comprises a user interface element to be displayed as part of the graphical user interface; associating at least one function with the defined user interface element; associating at least one playlist with the graphical user interface, wherein the playlist comprises a plurality of entries, and wherein each entry comprises a first media file or a pointer to the first media file; storing the graphical user interface and the association of the playlist with the graphical user interface in a manner interpretable by a web browser; and, associating computer program process code with the graphical user interface, wherein the computer program process code comprises instructions for determining whether a media player capable of playing the first media file is accessible to a computing device; causing the media player determined capable of playing the first media file to be loaded by the computing device, wherein at least controls of the media player are hidden from a user; monitoring for interaction with the defined user interface element; and, controlling the media player via the user interface element so as to cause the media player to function in a manner corresponding to the function associated with the user interface element.

[0011] In some embodiments, the graphical user interface may exist independent from any media player, and may comprise one or more graphics files. In some embodiments, the playlist may further comprise a media type indicator associated with the first media file, and the media player capable of playing the first media file may be determined, at least in part, by the media type indicator associated with the first media file. In some embodiments, the graphical user interface and the association of the computer program process code with the graphical user interface may be stored in a manner interpretable by a web browser. In some embodiments, the graphical user interface, the association of the graphical user interface and the playlist, and the association of the graphical user interface and the computer program process code are stored in the same file.

[0012] Some embodiments comprise a system for controlling a plurality of media players, comprising a defined graphical user interface, wherein the defined graphical user interface receives user interactions, wherein the defined graphical user interface comprises a user interface element, wherein the defined user interface element has a function associated therewith, and wherein the defined graphical user interface is stored
in a markup language derived from the Standardized Generalized Markup Language and interpretable by a web browser running on a computing device; a set of instructions, interpretable by the web browser, whereby the web browser can send commands to one of a plurality of media players installed on the computing device, thereby permitting the web browser to control playback of a media file based on user interactions with the defined graphical user interface; and a playlist, wherein the playlist is stored in a markup language derived from the Standardized Generalized Markup Language and interpretable by the web browser, wherein the playlist comprises a plurality of entries, and wherein at least a subset of the plurality of entries corresponds to a multimedia file.

[0013] In some embodiments, the set of instructions may further comprise instructions for determining whether a media player capable of playing a media file is installed on the computing device. In some embodiments, each playlist entry may further comprise at least one media type indicator associated with the playlist entry, wherein the determination of whether a media player capable of playing the media file is installed on the computing device is based, at least in part, on the media type indicator associated with a playlist entry. In some embodiments, the playlist and the graphical user interface are stored using a common markup language, such as, without limitation, the extensible Markup Language and/or the Hypertext Markup Language.

[0014] Some embodiments comprise a browser interpretable document comprising a playlist comprising a plurality of entries, wherein each entry comprises a first media file or a pointer to the first media file; a graphical user interface definition, or a pointer to the graphical user interface definition, the graphical user interface definition comprising a first user interface element and a function associated with the first user interface element; and, a set of commands, or a pointer to the set of commands, for controlling the operation of a plurality of media players; wherein, when the browser interpretable document is rendered by a conventional web browser, the web browser renders a graphical user interface based on the graphical user interface definition, and wherein the rendered graphical user interface controls one of the plurality of media players to facilitate playing of the first media file.
In some embodiments, the graphical user interface definition may be defined in a standardized markup language derived from the Standardized Generalized Markup Language, such as, without limitation, the Hypertext Markup Language. In some embodiments, the set of commands may comprise instructions in a scripting language. In some embodiments, each playlist entry may further comprise a media type indicator associated with the first media file, and the set of commands comprises instructions through which the browser can determine whether a media player capable of playing the first media file is installed on a computing device on which the browser is running, and wherein such determination is made, at least in part, based on the media type indicator associated with the first media file.

It is to be understood that both the foregoing general description and the following detailed description are exemplary and explanatory and are intended to provide further explanation of the disclosed browser interpretable document. Additional features and advantages of the invention will be set forth in the description which follows, and in part will be apparent from this disclosure, or may be learned by practice of the invention. The objectives and other advantages of the invention will be realized and attained by the structure particularly pointed out in this written description, including any claims contained herein and the appended drawings.

**BRIEF DESCRIPTION OF THE DRAWINGS**

The accompanying drawings, which are included to provide a further understanding of the disclosed browser interpretable document for controlling a plurality of media players, are incorporated in and constitute a part of this specification, illustrate various embodiments and, together with the description, serve to explain the principles of at least one embodiment of the disclosed browser interpretable document.

In the drawings:

Figure 1 is a block diagram illustrating an exemplary method through which a browser interpretable document can be used to control a plurality of media players.

Figure 2 is an exemplary browser interpretable document providing a playlist.
Figure 3 is an exemplary browser interpretable document providing a user interface through which a plurality of media players can be controlled.

Figure 4 is an alternative exemplary browser interpretable document providing a user interface through which a plurality of media players can be controlled.

Figure 5 is an alternative exemplary browser interpretable document which is being used to control a media player.

Figure 6 is an alternative exemplary browser interpretable document which is being used to control a media player.

Figure 7 is an alternative exemplary browser interpretable document, wherein the user interface provided by the browser interpretable document also provides access to a playlist.

Figure 8 is an alternative exemplary browser interpretable document, wherein the user interface provided by the browser interpretable document also provides a user interface through which a playlist can be edited.

Figure 9 is an alternative exemplary browser interpretable document, wherein the user interface provided by the browser interpretable document permits drag-and-drop reordering of playlist entries.

Figure 10 is an alternative exemplary browser interpretable document, wherein the user interface provided by the browser interpretable document after the playlist entries have been reordered.

Figure 11 is an alternative exemplary browser interpretable document, wherein a playlist entry has been added by dragging and dropping a URL to content.

Figure 12 is an alternative exemplary browser interpretable document which facilitates storing the playlist.

Figure 13 is an exemplary browser interpretable document written in HTML.

Figure 14 is an alternative exemplary browser interpretable document written in HTML.

Figure 15 is an alternative exemplary browser interpretable document written in HTML.
[0034] Figure 16 is a block diagram illustrating an exemplary method through which appropriate digital rights management licensing information can be obtained.

DETAILED DESCRIPTION

[0035] Reference will now be made in detail to embodiments of the disclosed browser interpretable document for controlling a plurality of media players, examples of which are illustrated in the accompanying drawings.

[0036] In an embodiment, a browser interpretable document is provided comprising a plurality of content entries and a set of instructions for controlling a plurality of media players. In an embodiment, to ensure that the browser interpretable document can be read and interpreted by a wide variety of browsers, the browser interpretable document leverages existing browser interpretable languages such as, without limitation, Hypertext Markup Language ("HTML"), extensible Markup Language ("XML"), Cascading Style Sheets ("CSS"), and the like, and deviates from the standardized definitions of such browser interpretable languages as little as possible. By way of example, without limitation, a browser interpretable document embodiment described herein introduces only two new keywords into the traditional browser vocabulary, "hTrack" and "timed", which are both used as class names. Classes are defined in the HTML 4+ specification.

[0037] In a traditional markup language, such as HTML, a link, or pointer to content, might be written as <a href="/media/example.mp3">Example Link</a>. When such a link is rendered by a browser, the Uniform Resource Locator ("URL") and related tags will be hidden from the user, and the user will simply see the words Example Link, but because of the browser's interpretation of the markup language, the user can click on or otherwise interact with the words to access the content referenced in the link. When the user interacts with the words, the browser attempts to load the file from the path specified in the URL, the process for which may include creating a local copy of the example.mp3. If the browser does not detect any browser interpretable instructions at or near the beginning of the file, the browser can attempt to determine whether the file is of a type that can be interpreted by any plug-ins or other enhancements to the browser. If the file can be interpreted by a plug-in, the plug-in is loaded by the browser and the file is loaded
by the plug-in. If the File is not of a format interpretable by the browser or a plug-in, the browser can pass the file information to the operating system. The operating system then determines an appropriate application, such as a media player, that should be loaded to handle the file based on information from the file. By way of example, without limitation, the operating system may look at the file's extension (i.e. the characters after the right-hand most period in the file name) to determine an appropriate application to be loaded. In the example provided above, since example.mp3 is a media file, the operating system can cause a locally-installed media player to load, and can send instructions to the locally-installed media player to play example.mp3. This behavior can be cumbersome and, as described above, can result in the loading of media players with interfaces with which the user is unfamiliar.

[0038] By modifying the link such that the browser recognizes the file as being of a track (i.e. by employing the "hTrack" class), the browser can be instructed to take alternative actions when the user interacts with the link. By way of example, without limitation, by changing the link above to read &lt;a href="/media/example.mp3" class="htrack">mp3 link&lt;/a&gt;, the browser can be instructed that example.mp3 is a track, and that it should therefore be handled according to alternative instructions contained in the browser interpretable document or referenced therein.

[0039] One or more such tracks can be combined together in a "playlist", or list of content to be played by an appropriate media player. In an embodiment, the order in which the tracks are played can be determined by the order in which the tracks appear within the playlist or other browser interpretable document. By way of example, without limitation, a browser interpretable document comprising the code listed below in Table 1 would cause first.mp3 to play first, followed by second.mp3.

```html
&lt;a href="/media/first.mp3" class="htrack">first song&lt;/a&gt;
&lt;a href="/media/second.mp3" class="htrack">second song&lt;/a&gt;
```

<table>
<thead>
<tr>
<th>TABLE 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

[0040] Although the order in which the tracks are listed within the browser interpretable document can be used to determine the play order, in an embodiment this play order can be altered by taking advantage of the tabindex attribute defined in the well-known HTML
4+ and/or XHTML specifications. Thus, by modifying the code listed in Table 1 to that listed below in Table 2, the browser can be instructed to play first.mp3 first, even though it is not the first content listed in the playlist.

```xml
<a href="/media/second.mp3" class="htrack" tabindex="2" >second song</a>
<a href="/media/first.mp3" class="htrack" tabindex="1" >first song</a>
```

TABLE 2

[0041] Although described herein as employing HTML, alternative embodiments, including those employing XML Shareable Playlist Format ("XSPF"), MPEG Audio Layer 3 URL ("M3U"), or Advanced Stream Redirector ("ASX"), are also envisioned. The instant disclosure's use of HTML-style URL's and markup language is intended for clarity, and should not be interpreted as limiting the browser interpretable document and/or the playlists to such a markup language.

[0042] Additional attributes can also be utilized to further streamline identifying an appropriate media player for a given media type. By way of example, without limitation, a "type" attribute can be employed. The type attribute can give an advisory hint as to the media type available via the link. Such a hint can allow the browser interpretable document or instructions associated therewith to initiate alternative procedures in the event the media type is not currently supported by the device on which the browser is running. In an embodiment, authors who use such an attribute can take responsibility for managing the risk that the media type specified is inconsistent with the content available via the link.

[0043] Another attribute that can be employed is the "title" attribute. In an embodiment, the title attribute can be used in a manner similar to the way in which the //playlist/trackList/track/title attribute is used in an XSPF playlist. That is, the title attribute can provide a hint about the content's title.

[0044] Still another attribute that can be employed is the image or "img" attribute. In an embodiment, the image attribute can be used in a manner similar to the //playlist/trackList/track/image attribute in an XSPF playlist. That is, the image attribute can provide a link to album art or other images associated with the content.
Yet another attribute that can be employed is the "free-text" element. In an embodiment, the free-text element can be used in a manner similar to the //playlist/trackList/track/annotation attribute in an XSPF playlist. That is, the free-text element can provide a comment or other information about the content.

When used in combination to create a playlist entry in a browser interpretable document, an exemplary link might resemble that of Table 3, below.

```
<a href="http://example.com/mp3" class="htrack" title="My Sharona" type="audio/mpeg">
  <img alt="The Knack album art" src="http://example.com/img/albumart/the_knack.gif" />
  don't deny you love this song
</a>
```

**TABLE 3**

In an effort to allow users to play content in their preferred media players, some content providers will make the content available in multiple media types or employing different attributes within a media type (e.g., different bit rates), and such content providers may prefer that playlist entries corresponding to content provided by them include pointers to at least a subset of the available forms of content. To permit such multiple pointers to be implemented according to the instant browser interpretable document, the hTrack keyword can be applied to a container element. This is illustrated below in Table 4.

```
<div class="htrack">
  Hello, hTrack World!
</div>
```

**TABLE 4**

Given that a single playlist entry can contain multiple playable links to the content, as well as other links, the question then becomes how to distinguish playable links from other links. By way of example, without limitation, in an hTrack container having multiple links such as that illustrated below in Table 5, there is no programmatic way to distinguish between the playable link "http://example.com/mp3" and one or more
links to other information, such as, without limitation, links to information about the artist and/or purchase the content to which the playable link pertains.

```
<div class="htrack">
  <a href="http://example.com/mp3">Hello, hTrack World!</a>
  <a href="http://example.com/about">Lyrics to this song</a>
</div>
```

### TABLE 5

[0049] The second new keyword, "timed", facilitates distinguishing between playable links and other, related links. The timed keyword expresses the idea that this is a URL where the timed media which is the core of this track can be found. It does not imply that there is only one such URL, and the presence of more than one timed media links is acceptable. An exemplary use of such a keyword is illustrated below in Table 6, and a more complete, HTML-based example is provided in Figure 15.

```
<div class="htrack">
  <a href="http://example.com/ogg" class="timed">Hello, hTrack World! (Ogg Vorbis)</a>
  <a href="http://example.com/qt" class="timed">Hello, hTrack World! (Quicktime)</a>
  <a href="http://example.com/about">Lyrics to this song</a>
</div>
```

### TABLE 6

[0050] Some skilled in the art will appreciate that the type attribute has functionality similar to enclosure elements for feed formats, thus the instant browser interpretable document may initially be seen as similar to technology underlying podcasting. However, upon more careful examination, the differences between podcasting and the instant browser interpretable document should be apparent. By way of example, the semantics of hTrack is a series of one or more timed media elements to be rendered in a user-set order. By contrast, the enclosure element used in podcasting facilitates pre-fetching of large files, reading entries in reverse chronological order, and copying media files to a portable device.

[0051] In conjunction with a playlist, a browser interpretable document according to the instant disclosure can also include a series of instructions for determining whether an appropriate media player is available for a given content's media type. In an
embodiment, such a series of instructions may be written in any programming or scripting language that facilitates calls to other applications, including languages such as, without limitation, AJAX, JAVA, JAVA Script, VB Script, C++, and the like. In an alternative embodiment, such a series of instructions may be written in any language, and may pass information to an intermediary "helper application" such as, without limitation, a browser plug-in or extension.

[0052] Where a playlist entry has a plurality of links to content (e.g., the same content at different bitrates or encoded using different media types), the instructions can also select an appropriate content link from those available based on attributes of the device on which the browser is operating. By way of example, without limitation, a playlist may include an entry for full motion video corresponding to a motion picture, and the playlist entry may include links to the video at different screen resolutions. The series of instructions can determine the device's display resolution and select from the set of links that which can most readily be rendered by the device without requiring unnecessary downloading and/or downsampling of higher resolution content.

[0053] In an embodiment, the series of instructions can also determine whether any of a plurality of standard media players are installed on the device. The series of instructions can then facilitate interaction with one or more of such media players through application programmer interfaces ("APIs") for those media players. In an embodiment, the series of instructions that facilitates access to such APIs is implemented as a separate module that can be readily updated, thereby permitting support for new media players and new media player versions to be easily added. Such updates can be obtained by polling a source at a regular interval (e.g., every time a browser interpretable document is loaded, once every week, every time the device restarts, etc.), or updates can be "pushed" to the device and/or browser.

[0054] An advantage of the browser interpretable document described herein is that, in addition to providing a well-structured playlist format that can be interpreted by and used across a wide variety of browsers, the browser interpretable document can also be used to define a user interface through which a plurality of media players can be controlled.
[0055] Figure 1 is a block diagram illustrating a method through which a browser interpretable document can be used to control a plurality of media players. In Block 100 of Figure 1, a graphical interface is created through which a plurality of media players are to be controlled. In an embodiment, such a graphical interface may be created as a single, flat image file, such as, without limitation, a JPEG-encoded image; a collection of textual elements, such as, without limitation, "<--", " <- " , " Stop ", " Play ", " || ", " - > ", " — > "; a collection of static and/or moving images; or combinations thereof. Collections of images and/or text can be combined into a browser interpretable document using HTML, XHTML, or other such languages to create the graphical user interface.

[0056] Figure 3 illustrates an exemplary embodiment in which a plurality of graphical and textual elements are combined in an HTML document to create such a graphical user interface. In Figure 3, buttons 310, 320, and 330 are intended to facilitate previous track, play/pause, and next track functionality, respectively. Button 340 provides a convenient link through which a user can access a content provider, in this case the well known Yahoo! music engine. Button 350 allows the user to view the playlist being controlled by buttons 310, 320, and 330.

[0057] In Block 110 of Figure 1, at least one user interface element, or control, is defined. In an embodiment, the user interface element may be defined by identifying one or more regions within an image, such as, without limitation, by using the map and area elements of HTML. In an embodiment, where the graphical user interface comprises one or more HTML elements, a container element, such as, without limitation, a <div> element, may be placed around one or more HTML elements, thereby identifying such HTML elements as a user interface element.

[0058] In Block 120, at least one function is associated with the defined user interface element. User interface elements are generally created to facilitate controlling or otherwise interacting with a media player, and in this step the specific functionality associated with the user interface element is defined.

[0059] In Block 130, at least one playlist is associated with the graphical user interface. The playlist comprises at least one media file or a pointer to the media file. In an embodiment, such a playlist may be defined using the playlist specification described
above, and in some embodiments, the playlist and the graphical user interface may be stored in the same file. Some embodiments may incorporate a reference to the graphical user interface into the playlist, and some embodiments may incorporate a reference to the playlist into the graphical user interface.

[0060] In Block 140, it is determined whether a media player capable of playing a media file in the playlist has been installed on the computing device on which the instant method is implemented. If no such media player has been installed, the device may be provided with a link through which an appropriate media player can be obtained (Block 150).

[0061] Once an appropriate media player has been identified and its installation status confirmed, the browser in which the instant method is implemented can cause the computing device to load the appropriate media player (Block 160). In an embodiment, the media player's native controls are hidden from the user (Block 170), and the user can utilize the controls provided by the graphical user interface to control the media player. In an embodiment, any images presented by the media player, such as, without limitations, static or dynamic images, visualizations, and the like, may be presented in a window, thereby facilitating the user's review of the same. In such an embodiment, the window may be affixed to or embedded in the graphical user interface. In an alternative embodiment, the instant method may permit the user to move the window containing the image. In Block 180, the instant method monitors the user's interactions with the user interface elements, and the commands associated with the user's interactions are sent to the loaded media player (Block 190).

[0062] Figure 2 is an exemplary browser interpretable document providing a playlist. In Figure 2, controls 200 facilitate controlling playback of each of content entries 210, 220, 230, 240, and 250. Content entry 210 is a link to content encoded using the MP3 media type. Content 220 is content encoded using the WMA media type. Content 220 is encoded using the Quicktime media type. Content 240 is encoded using the RealVideo media type. Content 250 is a link to content provided by a content provider, wherein the content is encoded using the WMA media type. Traditional media players attempting to play the content represented in this list would only be able to play a subset of the content,
because at least two of the media types are proprietary media types. As a result, the user would have to switch between media players to hear the represented content. The content also would not be playable in a continuous stream, as there is no communication between the media players that would facilitate playback initiation by the next media player. By contrast, by implementing a browser interpretable playlist in conjunction with the underlying series of instructions described above, user interface elements 200 can control each of the media players needed to play the content in this playlist, thereby providing a convenient, consistent user interface through which a plurality of media players can be controlled.

[0063] Figure 4 is an alternative exemplary browser interpretable document providing a user interface through which a plurality of media players can be controlled, wherein the browser interpretable document has been rendered by a browser. A text-based version of the corresponding browser interpretable file is provided in Figure 13. In this embodiment, the playlist has been defined according to the playlist format described above. Each playlist entry 400, 410, 420, and 430 includes at least one link to the content to be played, a link to album art or other graphic to be displayed, and a description of the content. Graphical user interface 440 is a window generated by a series of instructions referenced by the <script type="text/javascript" src="http://server.yahoo.com/user/file.js"></script> instruction in Figure 13. An exemplary set of instructions for generating such a graphical user interface and for receiving input from and interacting with a user via such an interface is included in Appendix A, which is incorporated herein by reference in its entirety. In an embodiment, such instructions can be written in Asynchronous JavaScript and XML ("AJAX") or other standardized language for producing interactive applications. The instructions can facilitate loading the graphical user interface, monitoring the user's interactions with the graphical user interface, and controlling a plurality of media players via the API's associated with the media players.

[0064] Figure 5 is an alternative exemplary browser interpretable document which is being used to control a media player. In Figure 5, the user has clicked button 510 to
move from playlist entry 410 to playlist entry 420, and graphical user interface 440 has
been updated to reflect the new content being played.

[0065] Figure 6 is an alternative exemplary browser interpretable document which is
being used to control a media player. In Figure 6, the user has right-clicked on graphical
user interface 440, thereby causing context-sensitive menu 610 to be displayed. In the
illustrated embodiment, context-sensitive menu 610 permits the user to control various
playback options, including repeating a playlist and randomly selecting from the content
defined in the playlist. Context-sensitive menu 610 also allows the user to indicate a
desire to view the playlist in a more streamlined interface. Such an interface is illustrated
in Figure 7.

[0066] Figure 7 is an alternative exemplary browser interpretable document, wherein the
user interface provided by the browser interpretable document also provides access to a
playlist interface 700. Playlist interface 700 can display a listing of the content
referenced in the playlist, and can also permit the user to exercise control over the playlist
and playlist interface 700. By way of example, without limitation, playlist interface 700
can include a plurality of user interface elements 710 through which the user can save the
playlist, undo edits made to the playlist, remove songs from the playlist, begin playback
of specific content from the playlist, hide the playlist interface, and the like. Playlist
interface 710 can also permit the user to edit the playlist, as illustrated in Figures 8-11.

[0067] In Figure 8, the user has clicked on or otherwise selected content entry 810. In
Figure 9, the user has begun to drag or otherwise reorder content entry 810 such that it
will appear in the playlist interface after content entry 820. In Figure 10, the user has
finished reordering the content entries in playlist 700 such that content entry 810 now
appears after content entry 820.

[0068] Figure 11 is an alternative exemplary browser interpretable document, wherein a
playlist entry has been added by dragging and dropping a URL to content from a separate
browser interface into playlist interface 700. In Figure 11, the user clicked on or
otherwise selected URL 1100, and dragged the URL into playlist interface 700. In the
illustrated embodiment, playlist interface 700 was able to obtain album art and other
information associated with URL 1100 based on the information contained in URL 1100.
Playlist interface 700 was then updated to include a reference to the underlying content behind URL 1100. In an embodiment, playlist interface 700 can also determine whether a user has the right to access the underlying content behind URL 1100, and if not, playlist interface 700 can facilitate obtaining the necessary rights.

[0069] Figure 12 is an alternative exemplary browser interpretable document which facilitates storing the playlist. As will be appreciated, when a user has altered the playlist, such as via playlist interface 700, the user may wish to save the altered playlist. In an embodiment, user interface 1200 can permit the user to save the altered playlist as a browser interpretable document.

[0070] In an embodiment, the URL to a particular content stream, or to particular content, may not be readily available. By way of example, without intending to limit the present invention, some services provide access to various forms of content by a variety of artists, as well as information about each piece of content. In such services, the publicly-available URL associated with a particular piece of content generally refers to the content by an ID number or name, such as, without limitation, 16472906. A user visiting the URL associated with the content, such as http://www.SomeServer.com/track/16472906 will typically see an information page that includes the artist's name, a description of the content, and the like, along with a button or other user interface element through which the content can be played. Although described herein as facilitating access to the entirety of the content, in alternative embodiments access may be limited to only a portion of the content.

[0071] In traditional media player environments, when the user clicks on the button or other user interface element, a media player is launched and the content is played. In such environments, the user is frequently unaware of any underlying URL's used to access the content. Because the publicly available URL to the content does not point directly to a content stream or a content file, it can be difficult for users to easily add the underlying content from such URL's to a playlist. However, in an embodiment, the browser interpretable document can include instructions for obtaining a URL to a content stream or content file corresponding to a public URL from such services. In an embodiment, the URL to the content stream or content file can be obtained by accessing
information from one or more web services. Similar means can be employed when the
user does not have access to particular content. By way of example, without limitation, a
particular playlist may include a reference to content stored on a remote server. If that
server is unavailable, the instant system can access copies available through alternative
content sources. Similarly, a playlist may include a reference to content for which the
user does not have appropriate permissions, and in an embodiment the instant system can
determine the underlying content or content stream and attempt to locate alternative
sources of the same content or content stream for the user. In an embodiment, the URL
for an underlying content or content stream can be obtained by requesting appropriate
information from a web service, such as, without limitation, the services provided by
MusicNet, using Simple Object Access Protocol ("SOAP"), Inter-ORB Protocol (HOP),
Remote Method Invocation ("RMI"), or other such means.

[0072] An exemplary method for obtaining a URL to underlying content or content
stream to which the user has appropriate DRM rights is illustrated in Figure 16. In Block
1600, login information, such as a login token, is acquired through which the user can be
identified to the content provider. A login token may be advantageous over a traditional
username/password combination, as such a token does not require that password be
transmitted. Although a login token is used in this exemplary embodiment, alternative
user authentication information, including traditional username/password combinations,
may be substituted therefor without departing from the spirit or the scope of the
invention. "Cookies", or information about the user, the browser, and/or the device, can
also be obtained from the browser (Block 1605). The login token and any required or
desired cookies are passed to the content provider, and a session is created through which
information can be communicated (Block 1610). In an embodiment, the session may be
encrypted, thereby facilitating secure communication between the browser and the
content provider.

[0073] The content provider can provide a token through which the browser can be
identified as corresponding to a properly authenticated user of the content provider's
system, wherein the token can be used to access information from one or more service
providers (Block 1615). In the illustrated embodiment, the browser can create a session
using the service provider token, through which the browser can communicate with a service provider (Block 1620). The browser and/or the content provider can supply information about the desired content, such as, without limitation, a reference number associated with the content, and the service provider can return a location URL through which the content can be accessed (Block 1625). The user's DRM rights are then queried to determine whether the user has appropriate rights to play the content (Block 1630). If the user has the appropriate DRM licensing information necessary to permit playback of the content, then the content is played (Block 1635).

[0074] If the user does not have appropriate DRM licensing information, an identifier specific to the device, browser, and/or user may be transmitted to the service provider to initiate the acquisition of such licensing information (Block 1640). In some embodiments, the content provider may act as a surrogate for some of the interactions between the user and the service provider, and the content provider and/or service provider can perform any transaction-related processes, such as, without limitation, billing the user for acquisition of the appropriate DRM rights.

[0075] In an embodiment, the service provider transfers appropriate DRM licensing information to the browser (Block 1645), and the browser stores this information (Block 1650). The DRM licensing information can be stored by the browser through a variety of means, including by using a Windows Media Player Digital Rights Management (WM-DRM) ActiveX control for the browser or other such controls. Once the URL and DRM licensing information have been obtained, the content can be played (Block 1655). In an embodiment, the URL supplied by the service provider may be a temporary URL, thereby facilitating validation and enforcement of the associated DRM licensing information each time the content is played.

[0076] Through the browser interpretable document described herein, users can easily create well-structured playlists, and play the content in those playlists through a single user interface, regardless of the media player responsible for playing the content. While detailed and specific embodiments of a browser interpretable document for controlling a plurality of media players have been described herein, it will be apparent to those skilled in the art that various changes and modifications can be made therein without departing
from the spirit and scope of a browser interpretable document for controlling a plurality of media players. Thus, it is intended that the present disclosure cover these modifications and variations provided they come within the scope of any appended claims and/or their equivalents.
YAHOO.ymu.wmpTypes = new Array();
YAHOO.ymu.wmpTypes ["text/plain"] = true;
YAHOO.ymu.wmpTypes ["video/ms-asf"] = true;
YAHOO.ymu.wmpTypes ["video/ms-wma"] = true;
YAHOO.ymu.wmpTypes ["video/ms-wmv"] = true;
YAHOO.ymu.wmpTypes ["video/x-ms-asf"] = true;
YAHOO.ymu.wmpTypes ["video/x-ms-wma"] = true;
YAHOO.ymu.wmpTypes ["video/x-ms-wmv"] = true;
YAHOO.ymu.wmpTypes ["audio/ms-asf"] = true;
YAHOO.ymu.wmpTypes ["audio/ms-wma"] = true;
YAHOO.ymu.wmpTypes ["audio/ms-wmv"] = true;
YAHOO.ymu.wmpTypes ["audio/x-ms-asf"] = true;
YAHOO.ymu.wmpTypes ["audio/x-ms-wma"] = true;
YAHOO.ymu.wmpTypes ["audio/x-ms-wmv"] = true;
YAHOO.ymu.wmpTypes ["application/ogg"] = true;
YAHOO.ymu.wmpTypes ["audio/mpeg"] = true;

////////////////////////////////////////////////////////////////////////////
///////////////////////
/soapRequest Functions
////////////////////////////////////////////////////////////////////////////

YAHOO.ymu.makeLaunchSoapRequest = function (request, operation, parameters,
mainRequest, callback)
{
    try
    {
        if (request == null;
        {
            request = YAHOO.ymu.createXMLHttpRequest ()
        }
    else
    {
        request . abort ();
    }

        request . onreadystatechange = function ()
        {
            if (request != null && (request . readyState == 4))
            {
                // SOAP Request Functions
                // declare your SOAP request, and
                // call the YAHOO.ymu.makeLaunchSoapRequest
                // function to send the request.
            }
        }
    }

    } catch (e) {
        console . log (e);
    }
}
// Prevent code from being re-entered:
var me = request;
request = null;
me.onreadystatechange = YAHOO.ymu.nullFunction,
callback. call (me);

// Set all parameters to null to ensure GC cleans up correctly:
me = null;
operaion = null;
parameters = null;
mamRequest = null;
callback = null;
}

request .open ("POST",
(mamRequest) "
"http://webservice.launch.yahoo.com/musicclient/mam.asrnx"
;
"http://webservice.launch.yahoo.com/musicclient/radio.asmx",
true);
request .setRequestHeader ("SOAPAction", "urn:yahoo:music:webservice/"
+ operation);
request. setRequest Header( "Content-type", "text/xml; charset=utf-8") ;
var soapBody = new Array ();
soapBody[0] = '<?xml version="1.0" encoding="utf-8"?>
<soap: Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd= "http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"""
xmlns:soap:"http://schemas.xmlsoap.org/soap/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd= "http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"""
xmlns:soap:"http://schemas.xmlsoap.org/soap/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd= "http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"""
xmlns:soap:"http://schemas.xmlsoap.org/soap/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd= "http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"""
xmlns:soap:"http://schemas.xmlsoap.org/soap/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd= "http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"""
xmlns:soap:"http://schemas.xmlsoap.org/soap/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd= "http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"""
xmlns:soap:"http://schemas.xmlsoap.org/soap/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd= "http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"""
xmlns:soap:"http://schemas.xmlsoap.org/soap/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd= "http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"""
xmlns:soap:"http://schemas.xmlsoap.org/soap/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd= "http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"""
xmlns:soap:"http://schemas.xmlsoap.org/soap/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd= "http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"""
xmlns:soap:"http://schemas.xmlsoap.org/soap/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd= "http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"""
xmlns:soap:"http://schemas.xmlsoap.org/soap/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd= "http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/""";
request .send (soapBody .join ("" ));
}
catch(e)
{
 if (request
={null)
{
 request .onreadystatechange = YAHOO.ymu,.null Function ;
 request = null;
}
callback. call (null);
}

return request;
}

YAHOO .ymu .makeMusicNetSoapRequest = function (request , operation, parameters, callback)
try {
  if (request == null) {
    request = YAHOO.ymu.createXMLHttpRequest();
  } else {
    request.abort();
  }

  request.onreadystatechange = function () {
    if ((request != null) && request.readyState == 4) {
      // Prevent code from being re-entered:
      var me = request;
      request.onreadystatechange = YAHOO.yrau.nFunction;
      callback.call(me);

      // Set all parameters to null to ensure GC cleans up correctly:
      me = null;
      operation = null;
      parameters = null;
      callback = null;
    }
  }

  request.open("POST", "http" + ((needHttps) ? "s" : "]") + "://stgapp.musicnet.com/"
    + /boss-net/services/FuIfllient", true);
  request.setRequestHeader("SOAPAction", "http://ws.musicnet.com/" + 
    operation);
  request.setRequestHeader("Content-type", "text/xml, charset=utf-8")
    var soapBody = new Array();
    soapBody[0] = '<?xml version="1.0" encoding="utf-8"?>
    <soap :Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsoap :BodyX = 
    soapBody[1] - operation;
    soapBody[3] = parameters;

    request.send(soapBody.join(""));
  }
catch (e) {
if (request != null)
{
    request.onreadystatechange = YAHOO.ymu.nullFunction;
    request = null;
}
callback.call(null);
}

return request;

YAHOO.ymu.DEFAULT_IMAGE = "http://us.entl.yimg.com/images.launch.yahoo cora/noPicture110.png";

YAHOO.ymu.getDataFromNode = function (nodeList, index, item)
{
    var xml = null;
    var dataNodeList;
    try
    {
        dataNodeList = nodeList.item(index).selectNodes(item);
        var length = dataNodeList.length;
        if (length > 0)
        {
            xml = dataNodeList.item(0).xml;
        }
    }
    catch (e)
    {
    }
    dataNodeList = null;
    return xml;
}

YAHOO.ymu/XMLFromNode = function (nodeList, index, item)
{
    var data = null;
    var dataNodeList;
    try
    {
        dataNodeList = nodeList.item(index).selectNodes(item);
        var length = dataNodeList.length;
        if (length > 0)
        {
            data = dataNodeList.item(0).firstChild.data;
        }
    }
catch (e)
{
  
  dataNodeList = null;

  return data;
}

YAHOO.ymu.getDataFromNodeOffset = function (nodeList, index, item, offset)
{
  var data = null;

  var dataNodeList;
  try
  {
    dataNodeList = nodeList.item(index).selectNodes(item);
    var length = dataNodeList.length;
    if (length > 0)
    {
      data = dataNodeList.item(Math.max(length-1-offset, 0)).firstChild.data;
    }
  }
  catch (e)
  {
  }
  dataNodeList = null;
  return data;
}

YAHOO.ymu.getElementsByTagName = function (nodeList, index, item)
{
  var value = null;

  var attrib;
  try
  {
    attrib = nodeList.item(index).attributes.getNamedItem(item);
    if (attrib != null)
    {
      value = attrib.value;
    }
  }
  catch (e)
  {
  }
  attrib = null;

  return value;
}

YAHOO.ymu.getImageSrcFromNode = function (nodeList, index, item, offset)
{
  var imageSrc = YAHOO.ymu.getDataFromNodeOfset (nodeList, index, item, offset);
}
return (imageSrc == null) ? imageSrc : YAHOO.ymu.DEFAULT_IMAGESRC;
}

////////////////////////////////////////////////////////////////////////////
///////////////////////
YMUSong
////////////////////////////////////////////////////////////////////////////
///////////////////////
YAHOO.ymu.YMUSong = function (songID, streamID, componentID, title, artist, album, trackNum, songURL, artistID, albumID, albumArtURL, sampleID)
{
    this.songID = songID || null;
    this.streamID = streamID || null;
    this.componentID = componentID || null;
    this.title = title || null;
    this.artist = artist || null;
    this.album = album || null;
    this.trackNum = trackNum || null;
    this.songURL = songURL || null;
    this.artistID = artistID || null;
    this.albumID = albumID || null;
    this.albumArtURL = albumArtURL || YAHOO.ymu.DEFAULT_IMAGESRC;
    this.sampleID = sampleID || null;
    this.idGUI = null;
}
YAHOO.ymu.YMUSong.prototype = {
    uiLogmCallback : null,
    setUILoginCallback : function (callback)
    {
        // Sets static UI function that returns MusicNet token
        YAHOO.ymu.YMUSong.prototype.uiLogmCallback = callback;
    },
    _onGetSongInfo : function (callback, ymuSong)
    {
        if (ymuSong !== null)
        {
            this.songID = ymuSong.songID;
            this.streamID = ymuSong.streamID;
            this.componentID = ymuSong.componentID;
            this.title = ymuSong.title;
            this.artist = ymuSong.artist;
            this.album = ymuSong.album;
            this.trackNum = ymuSong.trackNum;
            this.songURL = ymuSong.songURL;
            this.artistID = ymuSong.artistID;
            this.albumID = ymuSong.albumID;
            this.albumArtURL = ymuSong.albumArtURL;
            this.sampleID = ymuSong.sampleID;
        }
        callback.call(this);
    },
}
// Handler for YAHOO,ymu.YMUMusicNet.doLogin(); completion:_getSongInfoWithLogin: function (callback, isLoggedIn)
{
  var YMU = YAHOO.ymu;
  YMU.YMUMusicNet._getSongInfo (this, true, new
YMU.Callback (this._onGetSongInfo, this, callback));
},
onGetSongInfo_oUICallback: function (callback, attemptLogm)

  var YMU = YAHOO.ymu;
  if (attemptLogm)
  {
    YMU.YMUMusicNet.doLogin (YMU.YMULogm._getMusicNetToken (),
      YMU.YMULogin.getMachineID () ,
      new
YMU.Callback (this._getSongInfoWithLogin, this, callback));
  } else
  {
    YMU.YMUMusicNet.setClipPreviewState (true);
    this._getSongInfoWithLogin (callback, false);
  }
},
getSongInfo: function (getURL, callback)
{
  if (this.songID == null)
  {
    // Not a YMU track:
    callback.call(this);
  } else
  {
    var YMU = YAHOO.ymu;
    if (getURL)
    {
      this.songURL = null;
      var uiLogmCallback = this.uiLogmCallback;
      if (uiLogmCallback == null)
      {
        callback.call (this);
      } else
      {
        var YMUMusicNet = YMU.YMUMusicNet ;
        if (YMUMusicNet.getLogmState () M
YUMusicNet.getClipPreviewState ())
        {
          this._getSongInfoWithLogin (callback, true);
        } else
        {
          }
YMU.Callback.callPersist(new
YMU.Callback(this._onGetSongInfoUICallback, this, callbac k)),
}
} else
{
  YMU.YMUMusicNet.getSongInfo (this, false, new
YMU.Callback (this._onGetSongInfo, this, callback));
}
}
}

YAHOO.ymu.Playlist = function ()
{
  this.clear (),
}
YAHOO.ymu.Playlist.prototype =
{
  append: function (song)
  {
    this.list.push(song);
  },

  insert: function (index, song)
  {
    this.list.splice (index, 0, song);
  },

  move: function (oldIndex, newIndex)
  {
    var max = this.getListCount();
    if ( (oldIndex == newIndex) &&
    (oldIndex >= 0) &&
    (oldIndex < max) &&
    (newIndex < max))
    {
      var currentIndex = this.current;
      if (currentIndex == oldIndex)
      {
        this.current = newIndex;
      }
      else if ((currentIndex < oldIndex) &&
      newIndex) }
      else if ((currentIndex > oldIndex) &&
      newIndex) 
      {
        this.current++;
      }
      else if ((currentIndex <= oldIndex) &&
      newIndex) 
      {  

      }

  }
this.current = ;
}
var item = this.list.splice(oldIndex, 1),
this.list.splice(newIndex, 0, (item.length === undefined) ? item : item[0]));

remove: function (index)
{
    var max = this.getListCount();
    if ((index >= 0) && (index < max))
    {
        if (this.current > index)
        {
            this.current = ;
        }
        this.list.splice(index, 1);
        if (this.current >= th.s.list.length)
        {
            this.current = 0;
        }
    }
},
clear: function ()
{
    this.list = new Array();
    this.current = 0;
},
next: function ()
{
    this.current++;
    if (this.current >= this.list.length)
    {
        this.current = 0;
    }
},
previous, function ()
{
    this.current--;
    if (this.current < 0)
    {
        this.current = Math.max(0, th.s.isL.length - 1);
    }
},
isCurrentFirstInList : function ()
{
    return (this.current == 0);
isCurrentLastInList : function () {
    return (this .current >= (this .list .length - 1));
},

lsListEmpty: function () {
    return (this .list .length == 0);
},

getSongAt : function (index) {
    return ((index > 0) && (index < this .list .length)) ?
    this .list [index] : null;
},

getCurrentSong : function () {
    return (this .current < this .list .length) ? this .list [this .current ] : null;
},

setCurrentIndex : function (index) {
    var validIndex = ((index >= 0) && (index < this .list .length));
    if (validIndex) {
        this .current = index;
    }
    return validIndex;
},

getCurrentIndex: function () {
    return this .current;
},

getListCount : function () {
    return this .list .length;
},

destroy: function () {
    this .list = null;
}

YAHOO. ymu. Player = function ({

this. event = new YAHOO. uLil. CustomEvent ("player", this);
this .playlist = new YAHOO. ymu. P|aylist ();
this .currentURL = null;
this .stateRepeat = true;
this .stateShuf fie = false;

try
{
    this.wmp = new ActiveXObject ("WMPlayer .OCX");
    this.wmpControls = this.wmp .controls ;
    this.wmpSettings = this.wmp .settings ;
    this.wmpSettings .autoStart = true;
}
catch (e)
{
    this.wmpSettings = null;
    this.wmpControls = null;
    this.wmp = null;
}
YAHOO. ymu. Player . prototype =
{
    WMPPlayState enumeration values:
    WMPPLAYSTATE:
    {
    UNDEFINED : 0 ,
    STOPPED : 1 ,
    PAUSED : 2 ,
    PLAYING : 3 ,
    SCANFORWARD : 4 ,
    SCANREVERSE : 5 ,
    BUFFERING : 6 ,
    WAITING : 7 ,
    MEDIAENDED : 8 ,
    TRANSITIONING : 9 ,
    READY : 10 ,
    RECONNECTING : 11 ,
    LAST : 12
    },
    wmpPlayStateToText :
    [
    "Undefined",
    "Stopped",
    "Paused",
    "Playing",
    "Scanning Forward",
    "Scanning Backwards",
    "Buffering ...",
    "Waiting ...",
    "Media Ended",
    "Transitioning ...",
    "Ready",
    "Reconnecting"
    ],
    EVENTS:
Windows Media Player Cvent Handlers

_onCurrentItemChange : function (media)
{
    var EVENTS = this .EVENTS ;
    this .event .fire (EVENTS .GET_SONG_INFO | EVENTS .GET_CONTROL_STATE) ;
},

.onError: function ()
T
    var EVENTS = this .EVENTS ;
    this .event .fire (EVENTS .GET_SONG_INFO I EVENTS .GET_STATUS | EVENTS .REQUESTING_SONG | EVENTS .PLAYLIST_APPEND) ;
},

_onModeChange : function (modeName, newValue)
T
},

_onPlayStateChange: function (newState)
T
// set url : transitionmg->ready
// play: transitionmg->playing
// song ends: mediaEnded->Transitionmg->Stopped
var WMPPLAYSTATE = this .WMPPLAYSTATE ;

// If media has ended, skip to the next song if required:
if ((newState == WMPPLAYSTATE .MEDIAENDED) &&
    (this .playlist .isCurrentLastInList () ) M this .stateRepeat )
{
    var me = this ;
    setTimeout (function () {me .next (); me = null} , 1) ;
}

    if ((newState == WMPPLAYSTATE .READY) M (newState == WMPPLAYSTATE .STOPPED) ||
        (newState == WMPPLAYSTATE .PAUSED) M
        (newState == WMPPLAYSTATE .PLAYING) ) M (newState == WMPPLAYSTATE .MEDIAENDED) )
    {
        this .event .fire (this .EVENTS .GET_CONTROL_STATE) ;
    }
},

_onStatusChange : function ()

mitEvents : function!
{
    if (this.wmp != null)
    {
        var YMU = YAHOO.ymu;
        YMU.addScript("YAHOO_wmpevents.js", YMU.gooseRootURL + "wmpevents.js");
    }
},
addSubscriber : function (funcHandler, subscriber)
{
    this.event.subscribe(funcHandler, subscriber, true);
},
removeSubscriber : function (funcHandler, subscriber)
{
    this.event.unsubscribe(funcHandler, subscriber);
},
appendToPlaylist : function (song)
{
    this.playlist.append(song);
    this.event.fire(this.EVENTS.PLAYLIST_APPEND, song);
    this.event.fire(this.EVENTS.GET_CONTROL_STATE);
},
movePlaylistItem: function (oldIndex, newIndex)
{
    this.playlist.move(oldIndex, newIndex);
},
clearPlaylist : function()
{
    this.playlist.clear();
    this.stop();
    this.event.fire(this.EVENTS.GET_SONG_INFO | this.EVENTS.PLAYLIST_CLEAR);
},
getCurrentSong: function()
{
    return this.playlist.getCurrentSong();
},
getCurrentIndex : function()
return this.playlist.getCurrentIndex();
},

playSongAt: function (index)
{
    this.stop();
    if (this.playlist.setCurrentIndex(index))
    {
        this.play();
    }
},

removeSongAt: function (index)
{
    if ((index == this.playlist.getCurrentIndex()) &&
        (this.getPlayState() == this.WMPPLAYSTATE.PLAYING))
    {
        if (this.isNextAvailable())
        {
            this.next();
        }
        else
        {
            this.stop();
        }
    }
    this.playlist.remove(index);
},

getCurrentPlaylist: function ()
{
    return this.playlist;
},

_playGetSongInfoDone: function (song)
{
    if ((this.wmp !== null) && (song.songURL !== null) && (song ===
        this.getCurrentSong()))
    {
        try
        {
            this.wmp.URL = song.songURL;
            this.currentURL = song.songURL;
            this.event.fire(this.EVENTS.GET_CONTROL_STATE);
        }
        catch (e)
        {
        }
    }
},

play: function ()
{
    if (this.wmp !== null)
    {

try {
  var song = this.getCurrentSong();
  if (song !== null) {
    var EVENTS = this.EVENTS;
    if ((song.songURL !== null) && (this.currentURL !== song.songURL)) {
      var controls = this.wmpControls;
      if (controls !== null) {
        controls.play();
        this.event.fire(EVENTS.GET_CONTROL_STATE);
      }
      else {
        song.getSongInfo(true, new YAHOO.yimu.Callback(this._playGetSongInfo Donc, this));
        this.event.fire(EVENTS.REQUESTING_SONG);
      }
    }
  }
} catch (e) {
  }

pause: function () {
  try {
    var controls = this.wmpControls;
    if (controls !== null) {
      controls.pause();
      this.event.fire(this.EVENTS.GET_CONTROL_STATE);
    }
  } catch (e) {
  }
},

stop: function () {
  try {
    var controls = this.wmpControls;
    if ((this.wmp !== null) && (controls !== null)) {
      controls.stop();
      this.currentURL = "";
      this.wmp.URL = "";
    }
this.event.fire(this.EVENTS.GET_CONTROL_STATE);
}
catch (e)
{

}
next: function ()
{
    this.stop();
    this.playlist.next();
    this.play();
},
previous: function ()
{
    this.stop();
    this.playlist.previous();
    this.play();
},
getPlayState: function())
{
    var state = this.WMPPLAYSTATE.UNDEFINED;
    try
    {
        if (this.wmp !== null)
        {
            state = this.wmp.playState;
        }
    } catch (e)
    {
    }
    return state;
},
getStatus: function ()
{
    var status = "";
    try
    {
        if (this.wmp !== null)
        {
            status = this.wmp.status;
        }
    } catch (e)
    {
    }
    return status;
},
getDuration: function())
{
    var duration = 1;
try {
    if (this.wmp != null) {
        var media = this.wmp.currentMedia;
        if (media != null) {
            duration = media.duration;
        }
    }
} catch (e) {
}
return duration;
},
getDurationString : function () {
    var duration = "";
    try {
        if (this.wmp != null) {
            var media = this.wmp.currentMedia;
            if (media != null) {
                duration = media.durationString;
            }
        }
    } catch (e) {
    }
    return duration;
},
getTimeString: function () {
    return this.__getPlayerValue(this.wmp.controls, "currentPositionString" );
},
putPlayerValue : function (type, setting, value)
try {
    if (type != null) {
        type[setting] = value;
    }
} catch (e) {
}
}
getPlayerValue : function (type, setting)
T
var value = 0;
try
{
    if (type != null)
    {
        value = type[setting];
    }
}
catch (s)
{
    return value;
},

setSeek: function (position)
f
    // position: the time to seek to m seconds.
    this._putPlayerValue (this.wmpControls, "currentPosition", position);
},

getSeek: function ()
{
    return this._getPlayerValue (this.wmpControls, "currentPosition");
},

setVolume : function (volume)
{
    this._putPlayerValue (this.wmpSettings, "volume", Math.min (Math.max (0, volume), 100));
},

getVolume : function ()
{
    return this._getPlayerValue (this.wmpSettings, "volume");
},

setMute: function (mute)
{
    this._putPlayerValue (this.wmpSettings, "mute", mute);
},

getMute: function ()
{
    return this._getPlayerValue (this.wmpSettings, "mute");
},

setRepeat : function (repeat)
{
    this.stateRepeat = repeat;
},

getRepeat: function ()
{
    return this.stateRepeat;
},
setShuffle: function (shuffle)
{
    this.stateShuffle = shuffle;
},

getShuffle: function()
{
    return this.stateShuffle;
},

isControlAvailable : function (control)
{
    var available = false;
    try
    {
        var controls = this.wmpControls;
        if (controls != null)
        {
            available = controls.isAvailable(control);
        }
    }
    catch (e)
    {
    }
    return available;
},

isPlayAvailable: function ()
{
    return (!this.playlist.isListEmpty() &&
    this._isControlAvailable("play"));
},

isPauseAvailable : function ()
{
    return this._isControlAvailable("pause");
},

isStopAvailable : function ()
{
    return this._isControlAvailable("stop");
},

isNextAvailable : function()!
{
    return (this.playlist.getListCount() > 1);
},

isPreviousAvailable : function()!
{
    return (this.playlist.getListCount() > 1);
},

isSeekAvailable : function()!
{
    return this._isControlAvailable("currentPosition");
}
destroy: function ()
{
  if (this.playlist ' = null)
  {
    this.playlist. destroy ();
    this.playlist = null;
  }
  if (this.wmpControls ' = null)
  {
    this.wmpControls .stop ();
    this.wmpControls = null;
  }
  if (this.wmp ' = null)
  {
    this.wmp = null;
  }
  this.event = null;
  this.currentURL = null;
}
onLogmKey: function (theEvent)

    if (theEvent.keyCode == 13)
    {
        this.btnSignln.clicklt(true);
    }

onLogm: function (callback, loggedln)

    this.btnLogo.enable(true);
    this.btnPrev.enable(true);
    this.btnNext.enable(true);
    this.btnStop.enable(true);

    if (loggedln)
    {
        this._removeLoginPanel(),
        callback.call(true),
    }
    else
    {
        var UI = YAHOO.ymu_ui;
        var FILTER_ALPHA = "DXImageTransform.Microsoft.Alpha";
        for (var i in this.opacityElements)
        {
            var el = UI.getElementById(this.opacityElements[i]);
            if ((el == null) && (el.filters != null))
            {
                el.filters.item(FILTER_ALPHA).opacity = 100;
            }
        }
        var textID = UI.getElementById("YAHOO_ymu_player_login_ID");
        textID.disabled = false;
        UI.getElementById("YAHOO_ymu_player_login_password").disabled = false;
        this.btnSignIn.enable(true);
        this.btnRememberMe .enable (true);
        UI.setInnerText (UI.getElementById("YAHOO_ymu_player_login_status"), "Sign in failed. Please try again ");
        textID.focus ();

        // Show login status for 5 seconds and then replace it with button for 30-second clips,
        var me = this;
        this.logmStatusTimer = setTimeout (function ()
        {
            // Make sure login panel is still there:
            if ((me.logmStatusTimer != null) && (me.btn30Sec != null))
            {
            }
Ul.setInnerText (Ul.getClementById("YAHOO_ymu_player_logn_sLatus"), ");
me.btn30Sec. visible (true);
me.btn30Sec. enable (true);
me.loginStatusTimer = null;
me = null;
1, 5000
);
)
);

_onSigninButtonClick: function (callback)
{
var YMU = YAHOO.ymu;
var UI = YAHOO.ymu__ui;
var FILTER_ALPHA = "DXImageTransform.Microsoft.alpha";
var textID = UI.getElementById("YAHOO_ymu_player_logm_ID");
var textPassword = UI.getElementById("YAHOO_ymu_player_login_password");
textID, disabled = true;
textPassword . disabled = true;
this . btnSignln . enable (false);
if (this . loginStatusTimer != null)
{
    clearTimeout (this . loginStatusTimer) ;
    this . loginStatusTimer = null;
}
this.btn30Sec.enable(false);
this.btn30Sec.visible(false);
this . btnRememberMe . enable(false);
this . btnLogo . enable (false);
this. btnPrev. enable (false);
this . btnNext . enable (false);
this . btnStop . enable (false);
for (var i in this . opacityElements)
{
    var el = UI.getElementById (this . opacityElements [i]);
    if (!el != null) & & (el . filters != null)
    {
        el . filters . item (FILTER_ALPHA) . opacity = 40;
    }
}

UI.setInnerText (UI.getElementById ("YAHOO_ymu_player_login__status "), 
"Signing into Yahoo! Music...");

YMU.YMULogin .doLogin (text ID . value ,
textPassword . value,
this . btnRememberMe . getState(),
new YMU .Callback (this . _onLogi π , this,
callback) );
}
_on30SecButtonClick: function (callback)
{
    this._removeLoginPanel();
callback.call(false);
},

_showLoginPanel : function (callback)
{
    if (this._showmgLoginPanel())
        alert("Assertion failure in _showLoginPanel");
    this._removeLogmPanel();
}
this._showPlaylistPlayIcon();
if (this.player != null)
{
    var YMU = YAHOO.ymu;
    var UI = YAHOO.ymu_ui;
    var EVENT = YAHOO.util.Event;
    this.opacityElements = [
        "YAH00_ymu_player_login_instructions",
        "YAHOO_ymu_player_login_ID_caption",
        "YAHOO_ymu_player_login_ID",
        "YAH00_ymu_player_logm_password_caption",
        "YAH00_ymu_player_logm_username"
    ];

    var trackPanel = UI.getElementById("YAHOO_ymu_player__track Panel");
    if (trackPanel != null)
        trackPanel.style.display = "none";

    var loginPanel = UI.getElementById("YAHOO_ymu__player__loginpanel");
    if (loginPanel != null)
        loginPanel.style.display = "block";

    loginPanel.innerHTML = '<div id="YAHOO_ymu_player__logm_instructions" style="filter: progid:DXImageTransform.Microsoft.Alpha(opacity=100);">To play this track, sign into Yahoo ! Music</div>';
    
    <div id="YAHOO_ymu_player_login_ID_caption"
        style="filter: progid:DXImageTransform.Microsoft.Alpha(opacity=100);">Yahoo ID: </div>
    <input id="YAHOO_ymu_player_login_ID_bg" type="text"/>
    <div id="YAHOO_ymu_player_login_password_caption"
    <input id="YAH00_ymu_player_login_password_bg" type="password"/>
}
var textID = UI.getElementById("YAHOO_ymu_player_login_ID");
var textPassword = UI.getElementById("YAHOO_ymu_player_login_password");
EVENT.addListener(textID, "keydown", this._onLoginKey, this, true);
EVENT.addListener(textPassword, "keydown", this._onLoginKey, this, true);
this.btnRememberMe = new YUI.CheckButton("YAHOO_ymu_player_login_rememberme", "YAHOO_ymu_player_login_rememberme", true, "Remember My Yahoo' ID When Playing Music", "Remember My Yahoo' ID When Playing Music", null);
this.btnSignin = new YUI.Button("YAHOO_ymu_player_logm_signin", "YAHOO_ymu_player_login_signin", "Sign Into Yahoo! Music", new YMU.Callback(this._onSignmButtonClick, this, callback));
this.btn30Sec = new YUI.Button("YAHOO_ymu_player_login_30sec", "YAHOO_ymu_player_login_30sec", "Play 30-second clips", new YMU.Callback(this._on30SecButtonClick, this, callback));
textID.focus();
}
else{
callback.call(false);
}

_showingLoginPanel : function (){
var logm_panel = YAHOO.ymu_ui.getElementById("YAHOO_ymu_player_logmpanel");
return (logm_panel != null) && (logm_panel.style.display == "block");
}
removeLogmPanel: function (){
var UI = YAHOO.ymu_ui;
var EVENT = YAHOO.util.Event;
if (this.loginStatusTimer != null) {
    clearTimeout (this.logmStatusTimer);
    this.logmStatusTimer = null;
}

this.opacityElements = null;

if (this.btnRememberMe != null) {
    this.btnRememberMe.destroy();
    this.btnRememberMe = null;
}

if (this.btnSignln != null) {
    this.btnSignln.destroy();
    this.btnSignln = null;
}

if (this.btn30Sec != null) {
    this.btn30Sec.destroy();
    this.btn30Sec = null;
}

var login_panel = UI.getElementById("YAHOO_ymu_player_logmpanel");
if (login_panel != null) {
    var UI = YAHOO.ymu_ui;
    var EVENT = YAHOO.util.Event;

    EVENT.removeListener(UI.getElementById("YAHOO_ymu_player_logmpanel"),
                        "keydown", this._onLogmKey);
    EVENT.removeListener(UI.getElementById("YAHOO_ymu_player_logm_password"),
                        "keydown", this._onLogmKey);

    login_panel.innerHTML = "";
    login_panel.style.display = "none";
}

var track_panel = UI.getElementById("YAHOO_ymu_player__trackpanel");
if (track_panel != null) {
    track_panel.style.display = "block";
}
}

onLogmPersistAttempt: function (callback, loggedln)
{
    if (loggedln)
    {
        this.removeLogmPanel();
        callback.call(true);
    }
}
else
{
    this._showLoginPanel (callback);
}
}

login: function (callback)
{
    var YMU = YAHOO.ymu;
    YMU.YMULogm.doPersistLogin (new
    YMU.Callback (this._onLoginPersistAttempt, this, callback));
},

_updateControlState : function ()
{
    var player = YAHOO.ymu.Player;

    this.btnPlay.enable ((this.btnPlay.getStateO) ?
    player.isPauseAvailable () : player.isPlayAvailable ());
    this.btnPrev.enable (player.isPreviousAvailable ());
    this.btnNext.enable (player.isNextAvailable ());
    this.btnStop.enable (player.isStopAvailable ());
    this.thumbSeek.enable (player.isSeekAvailable ());
},

_setStatusTimeout : function (statusText, timeout, forceDisplay)
{
    var UI = YAHOO.ymu_ui;

    if (this.statusTimer != null)
    {
        clearTimeout (this.statusTimer);
        this.statusTimer = null;
    }

    UI.setInnerText (UI.getElementById ("YAHOO_ymu_player_status" ),
    statusText);

    if (timeout > 0)
    {
        this.forceStatusDisplay = forceDisplay;

        var me = this;
        this.statusTimer = setTimeout (function ()
        {
            me.forceStatusDisplay = false;
            if (me.statusTimer != null)
            {
                me._clearStatus ();
                me.statusTimer = null;
            }
            me = null;
        }, timeout);
    }
}
_clearStatus : function ()
{
    var player = YAHOO.ymu.Player;

    if (!((player.getPlayState() == player.WMPPLAYSTATE.PAUDED) && (this.playTimer == null)))
    {
        var UI = YAHOO.ymu_ui;
        UI.setInnerText (UI.getElementById("YAHOO_ymu_player_status").text);
    }
}

_onPlayTimer : function ()
{
    var player = YAHOO.ymu.Player;

    if ('this.forceStatusDisplay')
    {
        var UI = YAHOO.ymu_ui;
        var duration = player._getDurationString();
        var timestr = player._getTimeString();
        UI.setInnerText (UI.getElementById("YAHOO_ymu_player_status").text,
            (timestr == null) && (timestr.length > 0) ? (" */ " + duration) : ";
        }

        this.thumbsSelectors.setValue(player.getSeek());
    }
}

updateSongInfoDisplay : function ()
{
    var YMU = YAHOO.ymu;
    var UI = YAHOO.ymu_ui;
    var getElementById = UI.getElementById;
    var setInnerText = UI.setInnerText;

    var ymuSong = YMU.Player.getActiveSong();

    if (!((ymuSong == null) && ymuSong.songID))
    {
        getElementById ("YAHOO_ymu_player_song").innerHTML = '<a class="YAHOO_ymu_player_infolink" href="javascript:YAHOO.ymu.navigate("" + YMU.YMUSONGURLPREFIX + ymuSong.songID +'"')) + (ymuSong.title || "") + '</a>");
    }
    else
    {
        setInnerText (getElementById ("YAHOO_ymu_player_song"), (ymuSong == null) ? (ymuSong.title || "") : "");
    }

    if (!((ymuSong == null) && ymuSong.artistID))
    {
        getElementById ("YAHOO_ymu_player_artist") .innerHTML = 'O class="YAHOO_ymu_player_infolink" href="javascript:YAHOO.ymu.navigate("" +
YMUART I STURLPREFIX + ymuSong.artist ID + "') > + (y whole ymuSong.artist | | "") + '</a>';}
    else {
        setInnerText (getElementByld("YAHOO_ymu_player_artist" ), (y whole ymuSong = null) ? (ymuSong.artist | | "") : "");
    if ((ymuSong = null) && ymuSong.albumID) {
        getElementByld("YAHOO_ymu__player_album" ).innerHTML = '<a class="YAH00_ymu_player_mf ol ink' href=javascript:YAHOO, ymu.navigateC" + YMUART I STURLPREFIX + ymuSong.albumID + "') > + (ymuSong.album | | "") + ' </a>';}
    else {
        setInnerText (getElementByld("YAHOO_ymu_player_album" ), (ymuSong = null) ? (ymuSong.album | | "") : "");
    }
    var albumArt = gstElementByld("YAHOO_ymu_player_album_ar L");
    if ((albumArt = null) && (albumArt.filters = null))
    var filter = albumArt.filters.item("DXImageTransform. Microsoft. Alphalmage Loader"),
    filter.src = (ymuSong = null) ? ymuSong.albumArtURL : YMUART I STURLPREFIX + ymuSong.albumID + "') > + (ymuSong.album | | "") + ' </a>';}
    this._showPlaylist Playlcon ();
}

onPlayerEvent : function (eventType, args)
{
    var playerEvent = args[0];
    var YMU = YAHOO, ymu;
    var player = YMU.Player;
    var EVENTS = player. EVENTS;

    if (playerEvent & EVENTS.GET_SONG_INFO)
    { this._updateSongInf oDisplay ();
    }
    if (playerEvent & EVENTS.GET_CONTROL_STATE)
    { var playState = player.aplyState ();
        if (playState == player.WMPPLAYSTATE PLAYING)
            { this._thumbs eek. setRange (0, YMU.Player.getDuration ());
             }
            this.btnPlay.updateState (playState == player.WMPPLAYSTATE. PLAYING);
if (this.btnPlay, getState())
{
    if (this.playTimer != null)
    {
        clearInterval (this.playTimer);
    }
    var thisObject = this;
    this.playTimer = setInterval(function () {thisObject._onPlayTimer()}, 500);
}
else
{
   !if (this.playTimer != null)
    {
        clearInterval (this.playTimer);
        this.playTimer = null;
    }
}
this._updateControl State ()
;
}

if (playerEvent & EVENTS.REQU ESTING_SONG)
{
    this._setStatusTimeout ("Requesting song...", 0, false);
    this.thumbSeek.setValue (0);
    this.thumbSeek.enable (false);
}
else if (playerEvent & EVENTS.GET_STATUS)
{
    this._setStatusTimeout (player.getStatus ()), 2000, false);
}
if (this.playlist != null)
{
    if (playerEvent & EVENTS.PLAYLIST_CLEAR)
    {
        this._clearPlaylist();
    }

    if (playerEvent & EVENTS.PLAYLIST_APPEND)
    {
        this._appendPlaylist (args[1], true);  
    }
}

__onLogout : function (loggedln)
{
}

__onDrop: function ()
{
    var YMU = YAHOO.ymu;

    var song IDX = -1;
    var url = event.dataTransfer.getData ("Text");
    if (url != null)
function
{
  songIDX = url . mdexOf ('YMU . YMUSONCURLPREF1X');
  if (songIDX >= 0)
  {
    songID = url . substr (songIDX + YMU . YMUSONGURLPREFIX . length);
    var player = YMU . Player,
    player . appendToPlaylist (new YMU . YMUSong (songID, null, null, null, null, null, null, null, null, null));
    this . _setStatusTimeout ('Song added to playlist', 3000, true);
    if (player.isPlayAvailable () && 'this . _showLoginPanel ()
    {
      player . playSongAt (player . getCurrentPlaylistf) . getListCount ()
    - D ;
  } else
  {
    this . _setStatusTimeout ('No song was added', 3000, true);
  }
  return false;
  }

// Playlist GUI Methods
// _regDragDropListItem: function (list, item, group)
{
  var me = this;
  var id = item . id;

  var style = item . style;
  style . zIndex = 0;
  style . cursor = 'pointer';

  var dd = new YAHOO . util . DD (id, group);
  dd . onMouseDown = function (theEvent) (me . _onItemMouseDown (theEvent, id));
  dd . onMouseUp = function (theEvent) (me . _onItemMouseUp (theEvent, id));
  dd . onDragEnter = function (theEvent, objects) (me . _onItemDragEnter (theEvent, objects, id));
  dd . onDragOver = function (theEvent, objects) (me . _onItemDragOver (theEvent, objects, id));
  dd . onDragDrop = dd . onMouseUp;

  list . push (dd);
};

_unregDragDropList : function (list)
while (list . length > 0)
var dd = list.pop();

dd.onMouseDown = null;
dd.onMouseUp = null;

dd.onDragEnter = null;
dd.onDragOver = null;
dd.onDragDrop = null;
}

_clearPlaylistIDs : function!)

var playlist = YAHOO.yimu.Player.getCurrentPlaylist();
var length = playlist.getListCount();
for (var i = 0; i < length; i++)
{
    playlist.getSongAt(i).idGUI = null;
}

_getPlaylistIndex: function (itemId)
{
    var playlist = YAHOO.yimu.Player.getCurrentPlaylist();
    var length = playlist.getListCount();
    for (var i = 0; i < length; i++)
    {
        if (playlist.getSongAt(i).idGUI == itemId)
        {
            return i;
        }
    }
    return -1;
}

_createPlaylistItemButton: function (button, item, id, tooltip)
{
    if (button == null)
    {
        button.destroy();
    }

    var newBtn = YAHOO.yrau.ui.createElement("img");
    newBtn.id = id;
    newBtn.className = id;
    item.appendChild(newBtn);
    return new YAHOO.yimu.ui.Button(id, id, tooltip, new YAHOO.yimu.Callback(this._onPlayerButtonClick, this, id));
}

removePlaylistSelection : function ()
{
    if (this.deletePlaylistItemBtn == null)
    {
        this.deletePlaylistItemBtn.destroy();
        this.deletePlaylistItemBtn = null;
    }
}
if (this.playPlaylistItemBtn !== null)
{
    this.playPlaylistItemBtn.destroy();
    this.playPlaylistItemBtn = null;
}

if (this.playlistSelectedItem !== null)
{
    this.playlistSelectedItem.style.backgroundColor = "transparent";
    this.playlistSelectedItem = null;
}

_setPlaylistSelection: function (item)
{
    this._removePlaylistSelection();

    if (item !== null)
    {
        item.style.backgroundColor = "blue";

        this.deletePlaylistItemBtn = this._createPlaylistItemButton(this.deletePlaylistItemBtn, item, "YAHOO_ymu_playlist_delete", "Remove Song");
        this.playPlaylistItemBtn = this._createPlaylistItemButton(this.playPlaylistItemBtn, item, "YAHOO_ymu_playlist_play", "Play Song");

        this.playlistSelectedItem = item;
    }
}

_onPlaylistMouseDown : function (theEvent)
{
    this._removePlaylistSelection();
}

_onItemMouseDown : function (theEvent, source)
{
    source = YAHOO.ymu_ui.getElementByld(source);

    if (source !== null)
    {
        var style = source.style

        if (source !== this.playlistSelectedItem)
        {
            this.setPlaylistSelection (source);
        }

        YAHOO.util.Dom.setStyle (source, "opacity", ".5");
        style.zindex = 1;
        var me = this;
        this.playlistSetCursorTimer = setTimeout (function ()
        {
            if (me.playlistSetCursorTimer !== null)
            {
                style.cursor = "move";
            }
        }, 1000);
style = null;
me.playlistSetCursorTimer = null;
}
me = null;
}
);

this playlistDragDirection = null;
YAHOO.util.DragDropMgr.useCache = false;

_onItemClickMouseUp : function (theEvent, source)
{
    source = YAHOO.ymu.ui.getElementById (source);

    if (this.playlistSetCursorTimer != null)
    {
        clearTimeout (this.playlistSetCursorTimer);
        this.playlistSetCursorTimer = null,
    }

    if (source != null)
    {
        war style = source.style;
        YAHOO.util.Dom.setStyle (source, 'opacity', '1');
        style zIndex = 0;
        style cursor = 'pointer';
        style left = '0px';
        style top = '0px';
    }

    this.playlistDragDirection = null;
    YAHOO.util.DragDropMgr.useCache = true;

    _onItemClickDragOver : function (theEvent, objects, source)
    {
        if (this.playlistDragDirection != null)
        {
            var y = YAHOO.util.Dom.getY (source);
            if (this.playlistDragDirection)
            {
                if (y < this.playlistDragSourceY)
                {
                    this._onItemClickDragEnter (theEvent, objects, source);
                }
            }
            else
            {
                if (y > this.playlistDragSourceY)
                {
                    this._onItemClickDragEnter (theEvent, objects, source);
                }
            }
        }
    }

    this.playlistDragSourceY = y;
_onItemDragEnter : function (theEvent, objects, source) {
    var idSource = source;
    source = YAHOO.ymu_ui.getElementById(source);
    if (source != null)
    {
        var UI = YAHOO.ymu_ui;
        var DOM = YAHOO.util.DOM;
        this.playlistDragSourceX = DOM.getX(source);
        this.playlistDragSourceY = DOM.getY(source);
        var target = UI.getElementById(objects[0].id);
        var length = objects.length;
        for (var i = 0; i < length; i++)
        {
            var el = UI.getElementById(objects[i].id);
            if (DOM.getY(el) >= this.playlistDragSourceY)
            {
                target = el;
                break;
            }
        }
        if ((target != null) && (source != target))
        {
            var parent = source.parentNode;
            if (length > 1)
            {
                parent.insertBefore(source, target);
                this.playlistDragDirection = true;
            }
            else
            {
                this.playlistDragDirection = (DOM.getY(target) > this.playlistDragSourceY);
                if (this.playlistDragDirection)
                {
                    if (target == parent.firstChild)
                    {
                        parent.insertBefore(source, target);
                    }
                    else
                    {
                        var next = target.nextSibling;
                        if (next != null)
                        {
                            parent.insertBefore(source, next);
                        }
                        else
                        {
                            parent.appendChild(source);
                        }
                    }
                }
            }
        }
    }
}
else
  if (target == parent .lastChild)
  |
  |
else
  |
  |
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
song.songID + "\r\n";
} else {
    html += '<p>
        <a href="' + song.songURL + '"
            ';}
html += 'class="htrack"\r\ntitle="' + song.title + '
        "\r\ntype="audio/ms-wma"
        alt="' + song.album + '" src="' + song.albumArtURL + '
        ' + song.title + (song.artist ? ' - ' + song.artist : '') + '"
        >\r\n        ' + '\r\n        <a href="' + song.songURL + '"
            ';}
html += 'class="htrack"
        >\r\n        ' + song.title + (song.artist ? ' - ' + song.artist : '') + '"
        >\r\n        ' + '\r\n        <a href="' + song.songURL + '"
            ';}
html += '<script type="text/javascript" src="' + YMU.gooseRootURL + 'goose.js"></script>
    </body>
    </html>'
        return html;
            };
            onSavePlaylist : function () {
                if (this.savePlaylistDlg != null) {
                    var filename = YAHOO.ymu_ui.getElementById("YAHOO_ymu_playlist_savefilename").value;
                    if (filename) {
                        try {
                            if (this.fileSystemObject) this.GetExtensionName (filename) {
                                filename += "-html";
                            }
                            var i = filename.lastIndexOf (" "),
                                if (i < 0) {
                                    i = filename.length;
                                }
                                this.myMusicPath += "\\" + filename
                                this.fileSystemObject .CreateTextFile (this.myMusicPath,
                                    true),
                                var f this.fileSystemObject .GetFile (this.myMusicPath) ;
                                var ts = f.OpenAsTextLStream (2, -2);
                                ts.Write (this._convertPlaylistToHTML ("Playlist . " +
                                    filename .substr (0, i))) ;
                                ts.Close ;
                        } catch (e) {
                        }
                    }
                    this._onCloseSavePlaylist ();
                }
```javascript
_onCloseSavePlaylist : function ()
{
    if (this.savePlaylistDlg !== null)
    {
        this.savePlaylistDlgSelection = null;
        this.fileSystemObject = null;
        this.myMusicPath = null;
        this.savePlaylistDlg.hide();
        this.savePlaylistDlg.destroy();
        this.savePlaylistDlg = null;
    }
},

savePlaylist : function ()
{
    if (this.savePlaylistDlg === null)
    {
        var shell;
        list = '<ul id="YAHOO_ymu_playlist_savefile">'
        try
        {
            shell = new ActiveXObject("wscript.shell");
            this.myMusicPath = shell.RegRead("HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\My Music");
            this.fileSystemObject = new ActiveXObject("Scripting.FileSystemObject");
            var Z = this.fileSystemObject.GetFolder(this.myMusicPath);
            for (var fc = new Enumerator(Z.files), fc.atEnd();
             fc.moveNext();
            {
                var filename =
                this.fileSystemObject.GetFileName(fc.item());
                var extension =
                this.fileSystemObject.GetExtensionName(filename).toLowerCase();
                if ((extension == "html") || (extension == "htm"))
                {
                    list += '<li style="cursor:pointer">' + filename + '<br/>
                }
            }
        } catch(e)
        {
        }
        shell = null;
        list += "</ul>"
        this.savePlaylistDlg = new YAHOO.widget.SimpleDialog("YAHOO_ymu_playlist_savedialog",
        {
            visible : false,
            width: "300px"
        }
    }
```
effect: [{effect: YAHOO.widget.ContainerEffects.SLIDE, duration: 0.25},
  {effect: YAHOO.widget.ContainerEffects.FADE, duration: 0.25}]
};
this.savePlaylistsDlg.setHeader("Save Playlist As Web Page");
this.savePlaylistsDlg.setBody('<div id="YAHOO_ymu_playlist_savecaption">Playlists.</div>' + list + '<div style="text-align: center">Save as file name:<input type="text" style="margin-left: 8px;" id="YAHOO_ymu_playlist_savefilename"/>

var me = this;
this.savePlaylistsDlg.cfg.queueProperty(function() {
  (text: "Save"); handler: function() {me.__onSavePlaylistMouseDown(); me = null;}, isDefault: true
},
  (text: "Cancel"); handler: function() {me.__onCloseSavePlaylistMouseDown(); me = null;})
});

var listeners = new YAHOO.util.KeyListener(document,
  {keys: [27],
   fn: this.__onCloseSavePlaylist, scope: this,
   correctScope: true}
});
this.savePlaylistsDlg.cfg.queueProperty("keylisteners", listeners),
this.savePlaylistsDlg.newInstance().render(document.body);
this.savePlaylistsDlgSelection = null,
list = YAHOO.ymu_ui.getElementById("YAHOO_ymu_playlist_savefile"),
var children = list.childNodes,
var length = children.length;
for (var i = 0; i < length; i++)
  YAHOO.util.Event.addListener(children[i],
  "mousedown", this.__onSavePlaylistMouseDown, this, true);
this.savePlaylistsDlg.show();
setTimeout('YAHOO.ymu_ui.getElementById("YAHOO_ymu_playlist_savefilename").focus();', 1000);

__onSavePlaylistMouseDown: function (theEvent)
{
  YAHOO.util.Event.stopPropagation (theEvent);
}
if (this.savePlaylistDlgSelection !== null) {
    this.savePlaylistDlgSelection.style.background = "transparent";
    this.savePlaylistDlgSelection.style.color = "black";
}

this.savePlaylistDlgSelection = YAHOO.util.Event.getTarget(theEvent, true);
this.savePlaylistDlgSelection.style.background = "blue";
this.savePlaylistDlgSelection.style.color = "white";

YAHOO.ymu_ui.getElementById("YAHOO_ymu_playlist_filename").value = this.savePlaylistDlgSelection.innerHTML;
setTimeout('YAHOO.ymu_ui.getElementById("YAHOO_ymu_playlist_filename").focus()', 10);

clearPlaylist : function () {
    var playlist = YAHOO.ymu_ui.getElementById("YAHOO_ymu_playlist_list");
    if (playlist !== null) {
        var children = playlist.childNodes,
            for (var i = children.length - 1; i >= 0; i--)
                playlist.removeChild(children[i]);
        this._updatePlaylistScrollBar();
    }
    this._updateSongInfoDisplay();
    this._clearStatus();
}

onPlaylistInfoRequest : function (song) {
    this.pendingPlaylistInfoRequest = false;

    var UI = YAHOO.ymu_ui;

    var playlist = UI.getElementById("YAHOO_ymu_playlist_list");

    if ((song === null) && (playlist === null)) {
        if (song.artist === null) {
            // If we didn't get the metadata, try again.
            // Could happen if we try to get metadata while play engine
            // tries to get a URL at same time.
            this._addPlaylistInfoRequest (song);
            return;
        }

        this._setDisplayListItemInnerHTML (UI.getElementById(song ldGUI), song, true);
    }
this._checkPlaylistInfoRequest();

_addPlaylistInfoRequest : function (song)
{
  if ((song == null) && (song.songID == null) && (song.artist == null))
  {
    this.pendingPlaylistInfoSongs.push(song);
    this._checkPlaylistInfoRequest();
  }
},

_makePlaylistInfoRequest : function()
if (YAHOO.ymu == null) {
  return;
}
if ('YAHOO.ymu.YMUMusicNet.isInfoRequestBusy')
{
  this.pendingPlaylistInfoSongs.shift().getSongInfo(false, new YAHOO.ymu.Callback(this._onPlaylistInfoRequest, this)),
} else {
  var me = this;
  setTimeout (function()
  {
    me._makePlaylistInfoRequest();
    me = null;
  }, 1000
  );
}

checkPlaylistInfoRequest : function ()
if ('this .pendingPlaylistInfoRequest) && (this.pendingPlaylistInfoSongs.length > 0))
{
  this.pendingPlaylistInfoRequest = true;
  this._makePlaylistInfoRequest();
},

showPlaylistPlaylistIcon : function)
if (this.playlist == null)
{
  var UI = YAHOO.ymu_ui;
  var playlist = UI.getElementById('YAHOO_ymu_playlist_list ');
  if (playlist == null)
  {
    
    

62
var children = playlist.childNodes;
var count = children.length;
for (var i = 0; i < count; i++)
{
    children[i].firstChild.style.visibility = "hidden";
}

var song = YAHOO.ymu.Player.getCurrentSong();
if (song != null && song.idGUI)
{
    var element = UI.getElementById(song.idGUI);
    if (element != null)
    {
        element.firstChild.style.visibility = "visible";
    }
    else
    {
        alert("Assertion failure in _showPlaylistPlayIcon.
OK to continue.");
    }
}

_updatePlaylistScrollBar = function ()
{
    var scrollbar = this.playlistVScrollBar;
    if (scrollbar)
    {
        var container = YAHOO.ymu_ui.getElementById("YAHOO_ymu_playlist_list_container");
        var range = container.scrollHeight - container.offsetHeight + 1;
        scrollbar.setRange(range);
        scrollbar.show(range > 1);
    }
}

_setPlaylistItemInnerHTML = function (item, song, updatePlaylcon)
{
    var YMU = YAHOO.ymu;
    if ((song.songID) && (song.artistID))
    {
        item.innerHTML = '<div class="YAHOO_ymu_playlist_item_textlist"xli class="YAHOO__ymu_playlist"xli class="YAHOO__ymu_playlist_textlist"xli class="YAHOO__ymu_playlist_item_textlist"xli class="YAHOO__ymu_playlist_item_textlist"
    <a href="javascript:YAHOO.ymu.navigate("" + YMU.YMUSONGURLPREFIX + song.songID + ")">' + (song.title || "") + '</a>' + '<a href="javascript:YAHOO.ymu.navigate("" + YMU.YMUARTISTURLPREFIX + song.artistID + ")">' + (song.artist || "") + '</a></li></ul>";
}
else
{
    item.innerHTML = '<div class="YAHOO_ymu_playlist_item" x/div><img class="YAHOO_ymu_playlist_albumart" src="' + (song.albumArtURL || '') + '"

    clasa="YAHOO_ymu_playlist_item_pla con" x/div><img class="YAHOO_ymu_playlist_albumart" src="' + (song.albumArtURL || '') + '"

    clasa="YAHOO_ymu_playlist_item"

    if (updatePlaylcon)
    {
        this._showPlaylistPlayIcon();
    }
    this._updatePlaylistScrollBar();
    
    _append Playlist: function(song, bπnglntoView)
    {
        if ((song != null) && (this.playlist != null))
        {
            var UI = YAHOO.ymu ui;
            var playlist = UI.getElementById("YAHOO_ymu_playlist_list");
            var item = UI.createElement("li");
            song.idGUI = YAHOO.util.Dom.generateId();
            item.id = song.idGUI;
            item.className = "YAHOO_ymu_playlist_item";
            playlist.appendChild(item);
            this._setPlaylistItemInnerHTML(item, song, bπnglntoView);
            this._regDragDropListItem (this.playlistDDList, item,
            "YAHOO_ymu_playlist_group")
            ;
            this.addPlaylistlnfoRequest(song);
            if (bπnglntoView)
            {
                this.playlistVScrollBar .setPos (item.offsetTop);
            }
        },
    },

    fillPlaylist : function()
    {
        var playlist = YAHOO.ymu Player .getCurrentPlaylist();
        if ((this,playlist != null) & & (playlist != null))
        {
            var count = playlist.getListCount();
            for (var i = 0; i < count; i++)
            {
                this._appendPlaylist (playlist .getSongAt (i), false);
            }
            this._showPlaylistPlayIcon();
        },
    }

    _scrol!Playlist : function(pos)
togglePlaylist : function ()
    if (this.playlist == null)
    {
        try
        {
            var YMU = YAHOO.ymu;
            var UI = YAHOO.ymu_ui;
            this.ongmalDDMMode = YAHOO.util.DDM.mode;
            YAHOO.util.DDM.mode = YAHOO.util.DDM.INTERSECT;
            this.playlistDDLList = new Array();
            this.pendingPlaylistlnfoRequests = false;
            this.pendingPlaylistlnfoSongs = new Array();
            this.playlist = new YAHOO.widget.Panel("YAHOO_ymu_playlist",
                {effect: YAHOO.widget.ContainerEffect.TADE, duration: 0.25},
                width: "333px",
                height: "298px",
                fixedcenter: false,
                constrainmovviewport: false,
                underlay: "none",
                close: false,
                visible: false,
                draggable: true,
                modal: false
            );
        }
        this.playlist.setHeader('<div><div id="YAHOO_ymu_playlist_list_item"/>
      </div></div>');
        var container = UI.getElementById("YAHOO_ymu_playlist_list_item");
        // Need to get width of scrollbar and height of button
        this.playlistVScrollBar = new
        YMU.Callback(this.scrollPlaylist, this),
    }
    }
false);  

this._fillPlaylist ();  

var playlistHeader = this.playlist.header;  
YAHOO.util.Dom.setStyle (playlistHeader, "cursor", "auto");  

playlistHeader.onDragEnter = new Function ("return false");  
playlistHeader.onDragOver = playlistHeader.onDragEnter;  
playlistHeader.onDrop = new Function ("YAHOO.ymu.GUI._onDrop O");  
YAHOO.util.Event.addListener (playlistHeader, "mousedown", this._onPlaylistMouseDown, this, true);  

this.playlistSave = new UI.Button ("YAHOO_ymu_playlist_save",  
"YAHOO_yrau_playlist_save", "Save Playlist As Web Page",  
new YMU.Callback (this._onPlayerButtonClick, this, "YAHOO_ymu_playlist_save"));  
this.playlistClear = new UI.Button ("YAHOO_ymu_playlist_clear",  
"YAHOO_ymu_playlist_clear",  
"Clear All Songs",  
new YMU.Callback (this._onPlayerButtonClick, this, "YAHOO_ymu_playlist_clear"));  
YMU.Callback (this._onPlayerButtonClick, this, "YAHOO_ymu_playlist_close");  

this.playlist.show ();  
this.playlist.center ();  

var player = UI.getElementById ("YAHOO_ymu_player");  
var playlist = UI.getElementById ("YAHOO_ymu_playlist");  
if ((player != null) && (playlist != null)) {  
    player = player.offsetParent;  
    playlist = playlist.offsetParent;  
    if (((playlist.offsetLeft <= player.offsetLeft) &&  
            (playlist.offsetLeft + playlist.offsetWidth) >=  
            player.offsetLeft + player.offsetWidth) &&  
            (playlist.offsetTop <= player.offsetTop) &&  
            (playlist.offsetTop + playlist.offsetHeight) >=  
            player.offsetTop + player.offsetHeight)) {  
        this.playlist.moveTo (player.offsetTop + 50,  
            player.offsetTop + 50);  
    }  
}  
this._showPlaylistPlaylistIcon ();  

this.playlistSetCursorTimer = null;  
this.playlistSelectedItem = null;  
this.playlistDragDirection = null;  
this.deletePlaylistItemBtn = null;
this.playlistItemBtn = null;
}
catch(e)
{
}
}
else
{
this._onCloseSavePlaylist();

// Remove idGUI property from Player's playlist:
this._clearPlaylistIDs();

var playlistHeader = this.playlist.header;
playlistHeader.ondragenter = null;
playlistHeader.ondragover = null;
playlistHeader.ondrop = null;
YAHOO.util.Event.removeListener(playlistHeader, "mousedown", this._onPlaylistMouseDown);

this.pendmgPlaylistInfoRequest = null;
this.pendmgPlaylistInfoSongs = null;

YAHOO.util.DDM.mode = this.origDMMMode;
this.origDMMMode = null;
if (this.playlistSetCursorTimer != null)
{
clearTimeout(this.playlistSetCursorTimer);
this.playlistSetCursorTimer = null;
}
this._unregDragDropList(this.playlist.DDList);
this.playlistDDLList = null;
this.playlistSelectedItem = null;
this.playlistDragDirection = null;

if (this.deletePlaylistItemBtn != null)
{
this.deletePlaylistItemBtn.destroy();
this.deletePlaylistItemBtn = null;
}
if (this.playPlaylistItemBtn != null)
{
this.playPlaylistItemBtn.destroy();
this.playPlaylistItemBtn = null;
}

if (this.playlistSave != null)
{
this.playlistSave.destroy();
this.playlistSave = null;
}
if (this.playlistClear != null)
{
this.playlistClear.destroy();
this.playlistClear = null;
}
if (this.playlistClose != null)
{
    this.playlistClose.destroy();
    this.playlistClose = null;
}

if (this.playlistVScrollBar != null)
{
    this.playlistVScrollBar.destroy();
    this.playlistVScrollBar = null;
}

this.playlist.destroy();
this.playlist = null;

///////////////////////////////////////////////////////////////////////////
// End Playlist GUI Methods
///////////////////////////////////////////////////////////////////////////

_onMenuClick: function (item)
{
    var YMU = YAHOO.ymu;
    var player = YMU.Player;
    switch (item)
    {
        case 0:
            YMU.YMUMusicNet.doLogout (new YMU.Callback (this._onLogout, this));
            YMU.YMULogm.doLogout (true, new YMU.Callback (this._onLogout, this));
            break;
        case 1:
            player.setRepeat ('player.getRepeat ());
            break,
        case 2:
            player.setShuffle ('player.getShuffle ()');
            break;
        case 3:
            this.togglePlaylist ();
            break;
    }
    this.logoMenu.destroy ();
    this.logoMenu = null;
},

_onPlayerButtonClick : function (buttonID, state)
{
    var YMU = YAHOO.ymu;
    var UI = YAHOO.ymu_ui;
    switch (buttonID)
    {
        case "YAHOO_ymu_player_logobtn ",

if (this.logoMenu == null)
{
    this.logoMenu.destroy();
}
this.logoMenu = new UI.PopupMenu([{
    text: "Sign Out",
    checked: false},
    YMU.Player.getRepeat ()},
    {text: "Repeat"},
    YMU.Player.getShuffle ()},
    {text: "Shuffle"},
    this.playlist != null]);
this.logoMenu.show(this.btnLogo, new YAHOO.ymu.Callback(this._onMenuClick, this));
    break;
    case "YAHOO_ymu_player_prevbtn ":
    this._removeLoginPanel ();
    YMU.Player.previous ();
    break;
    case "YAHOO_ymu_player_playbtn" :
                if (state)
    {
        YMU.Player.play ();
    }
            else
    {
        YMU.Player.pause();
    }
            break;
    case "YAHOO_ymu_player_nextbtn" :
    this._removeLoginPanel ();
    YMU.Player.next ();
    break;
    case "YAHOO_ymu_player_stopbtn" :
    this._removeLoginPanel ();
    YMU.Player.stop ();
    break;
    case "YAHOO_ymu_player_mutebtn" :
    YMU.Player.setMute (state);
    break;
    case "YAHOO_ymu_playlist_save" :
    this._savePlaylist ();
    break;
    case "YAHOO_ymu_playlist_clear" :
    YMU.Player.clearPlaylist();
    break;
    case "YAHOO_ymu_playlist_close" :
                if (this.playlist != null)
    {
        this._togglePlaylist();
    }
            break;
    case "YAHOO_ymu_playlist_delete" :
    var selectedItem = this.playlistSelectedItem;
    if (selectedItem != null)
    {
        var index = this._getPlaylistIndex(selectedItem.id);
if (index >= 0) {
    selectedItem.parentNode.removeChild(select element);
}

var player = YAHOO.ymu.Player;
var haveCurrentLoginUp = ((index == player.getCurrentIndex()) & this._showingLoginPanel);
if (haveCurrentLoginUp) {
    this._removeLoginPanel();
}

player.removeSongAt(index);
this._showPlaylistPlayIcon();
this._updatePlaylistScrollBar();

var playlist = player.getCurrentPlaylist();
var listCount = playlist.getListCount();
if (index >= listCount) {
    index = 0;
}

var nextSong = playlist.getSongAt(index);
this._setPlaylistSelection((nextSong != null) ? YAHOO.ymu.ui.getDocumentElement(nextSong.idG) : null);

if (listCount > 0) {
    if (haveCurrentLoginUp) {
        // If login screen was up for the deleted item, force the player to play the next track as it would not do so automatically because the song was not actually playing yet.
        player.play();
    }
} else {
    // Empty playlist, update player display:
    this._updateSongInfoDisplay();
    this._clearStatus();
}

break;
}
case "YAHOO_ymu_playlist_play" :
    if (this.playlistSelectedItemId != null) {
        var index = this._getPlaylistIndex(this.playlistSelectedItemId.id);
        if (index >= 0) {
            this._removeLoginPanel(),
            YAHOO.ymu.Player.playSongAt (index);
        }
    }
break;
onPlayerSeek : function (thumbID, value)

YAHOO.ymu.Player.setSeek (value);

onPlayerVolume : function (thumbID, value)

YAHOO.ymu.Player.setVolume (value);

createPlayer : function)

var YMU = YAHOO.ymu;
var UI = YAHOO.ymu.ui;
var EVENT = YAHOO.util.Event;

try

this.player = new YAHOO.widget.Panel("YAHOO_ymu_player",

{effect: YAHOO.widget.ContainerEffect.fade, duration:0.25},

width: "333px",

height: "222px",

fixedcenter: false,

constraintviewport: false,

underlay: "none",

close: false,

visible: false,

draggable: true,

modal: false

});

this.player.setHeader("<div><div id="YAHOO_ymu_player_trackbar" style="display:block;" XUL
id="YAHOO_ymu_player_menu" <li id="YAHOO_ymu_player_status" style="cursor:default">X</li>
li id="YAHOO_ymu_player_artist" style="cursor:default">X</li>
li id="YAHOO_ymu_player_album" style="cursor:default">X</li>
li id="YAHOO_ymu_player_tlpb"
width="333px",
height="222px",
fixedcenter: false,
constraintviewport: false,
underlay: "none",

close: false,

visible: false,

draggable: true,

modal: false

});

YAHOO.util.Dom.setStyle(playerHeader, "cursor", "auto");
playerHeader .ondragenter = new Function ("return false");
playerHeader .ondragover = playerHeader .ondragenter;
playerHeader .ondrop = new Function ("YAHOO.ymu.GUI._onDrop()");
this.btnLogo = new UI.Button ("YAHOO_ymu_player_logoBtn", "Options", new YMU.Callback (this._onPlayerButtonClick, this, "YAHOO_ymu_player_logoBtn"));
this.btnPrev = new UI.Button ("YAHOO_ymu_player_prevbtn", "Previous", new YMU.Callback (this._onPlayerButtonClick, this, "YAHOO_ymu_player_prevbtn"));
this.btnPlay = new UI.CheckButton ("YAHOO_ymu_player_playbtn", "Play", "Pause", new YMU.Callback (this._onPlayerButtonClick, this, "YAHOO_ymu_player_playbtn"));
this.btnNext = new UI.Button ("YAHOO_ymu_player_nextbtn", "Next", new YMU.Callback (this._onPlayerButtonClick, this, "YAHOO_ymu_player_nextbtn"));
this.btnStop = new UI.Button ("YAHOO_ymu_player_stopbtn", "Stop", new YMU.Callback (this._onPlayerButtonClick, this, "YAHOO_ymu_player_stopbtn"));
this.btnMute = new UI.CheckButton ("YAHOO_ymu_player_mutebtn", "Mute", "Restore Sound", new YMU.Callback (this._onPlayerButtonClick, this, "YAHOO_ymu_player_mutebtn"));
this.thumbSeek = new UI.Thumb ("YAHOO_ymu_player_seekthumb", "Seek", 0, YMU.Player.getDuration (), new YMU.Callback (this._onPlayerSeek, this, "YAHOO_ymu_player_seekthumb"));
this.thumbVolume = new UI.Thumb ("YAHOO_ymu_player_volumethumb", "Volume", 07 100, new YMU.Callback (this._onPlayerVolume, this, "YAHOO_ymu_player_volumethumb"));
this.player.show ();
this.player.center O;
this.thumbSeek.setValue (YMU.Player.getSeek ());
this.thumbVolume.setValue (YMU.Player.getVolume ());
this._updateControlState();
}
catch (e)
{
    YMU.print(YMU.message(e));
}
},
destroyPlayer: function()
{
    if (this.playTimer !== null)
    {
        clearInterval(this.playTimer);
        this.playTimer = null;
    }
    if (this.statusTimer !== null)
    {
        clearTimeout(this.statusTimer);
        this.statusTimer = null;
    }
    this._removeLogmPanel();
    this.forceStatusDisplay = null;
    if (this.playlist !== null)
    {
        this.__togglePlaylist();
    }
    if (this.logoMenu !== null)
    {
        this.logoMenu.destroy();
        this.logoMenu = null;
    }
    if (this.btnLogo !== null)
    {
        this.btnLogo.destroy();
        this.btnLogo = null;
    }
    if (this.btnPrev !== null)
    {
        this.btnPrev.destroy();
        this.btnPrev = null;
    }
    if (this.btnPlay !== null)
    {
        this.btnPlay.destroy();
        this.btnPlay = null;
    }
    if (this.btnNext !== null)
    {

```javascript
this.btnNext.destroy();
this.btnNext = null;
}

if (this.btnStop !== null) {
    this.btnStop.destroy();
    this.btnStop = null;
}

if (this.btnMute !== null) {
    this.btnMute.destroy();
    this.btnMute = null;
}

if (this.thumbSeek !== null) {
    this.thumbSeek.destroy();
    this.thumbSeek = null;
}

if (this.thumbVolume !== null) {
    this.thumbVolume.destroy();
    this.thumbVolume = null;
}

var playerHeader = this.player.header;
playerHeader.onDragEnter = null;
playerHeader.onDragOver = null;
playerHeader.onDrop = null;

if (this.player !== null) {
    this.player.hide();
    this.player = null;
}
```

…
Instantiate objects.

```javascript
UI.imtModule();

YMU.GUI = new YMU.GUI();

(new YMU.YMUSong()).setUILoginCallback(
    new YMU.Callback(YMU.GUI.login, YMU.GUI));

YMU.YMULogin = new YMU.YMULogin();

YMU.YMUMusicNet = new YMU.YMUMusicNet();

YMU.Player = new YMU.Player();

YMU.Player.initEvents();

YMU.GUI.createPlayer();

YMU.Player.addSubscriber(YMU.Player.Event, YMU.GUI);

function returnTrue() { return true; }

var SONGIDPARAM = "?songid=";

var length = SONGIDPARAM.length;

var player = YMU.Player;

var playlist = YMU.parsePlaylist();

var length = playlist.length;

for (var i = 0; i < length; i++) {
    var e = playlist[i];
    var href = e.href.toLowerCase();
    var songID = null;

    if (e.protocol.toLowerCase() == "ymp:"
```
$('title, null, albumTitle, null, href, null, null, albumArt URL, null));
}
player.play();

YAHOO.ymu.onUnload = function (e) {
  var YMU = YAHOO.ymu;
  if (YMU.GUI !== null) {
    if (YMU.Player !== null) {
      YMU.Player.removeSubscriber(YMU.GUI.onPlayerEvent, YMU.GUI);
    }
    YMU.GUI.destroyPlayer();
  }
  if (YMU.Player !== null) {
    YMU.Player.destroy();
    YMU.Player = null;
  }
  if (YMU.YMUMusicNet !== null) {
    YMU.YMUMusicNet.destroy();
    YMU.YMUMusicNet = null;
  }
  if (YMU.YMULogin !== null) {
    YMU.YMULogin.destroy();
    YMU.YMULogin = null;
  }
}

YAHOO.ymu.checkScriptsLoaded = function () {
  if (YAHOO.util.Dom !== undefined) && (YAHOO.util.CustomEvent !== undefined) && (YAHOO.util.Amm !== undefined) && (YAHOO.util.DragDrop !== undefined) && (YAHOO.ymu_ui !== undefined) && (document.getElementById('YAHOO_container_css').readyState === "complete") && (document.getElementById('YAHOO Goose_css').readyState === "complete") {
    if (document.getElementById('YAHOO_container_2') !== null)
      YAHOO.ymu.addScript("YAHOO_container_2",
        "http: //us.js2.yimg.com/us.js.yimg.com/l;b/common/http://s/2/container/ner/container_2.0.1-b4.3.s"');
  }
}.
if (YAHOO.widget.Panel = undefined) {
    // All scripts are loaded... call main:
    setTimeout("YAHOO, ymu.mam()", 1);
    return;
}

setTimeout("YAHOO, ymu.checkScriptsLoaded()", 100);

YAHOO, ymu.checkScriptsLoaded(); // Calls mam() when all scripts are loaded
[0077] What is claimed is:

1. A method of controlling a plurality of media players using a standardized interlace, comprising:
   - defining a graphical user interface through which a plurality of media players are to be controlled, wherein the graphical user interface comprises a user interface element to be displayed as part of the graphical user interface;
   - associating at least one function with the defined user interface element;
   - associating at least one playlist with the graphical user interface, wherein the playlist comprises a plurality of entries, and wherein each entry comprises a first media file or a pointer to the first media file;
   - storing the graphical user interface and the association of the playlist with the graphical user interface in a manner interpretable by a web browser; and,
   - associating computer program process code with the graphical user interface, wherein the computer program process code comprises instructions for:
     - determining whether a media player capable of playing the first media file is accessible to a computing device;
     - causing the media player determined capable of playing the first media file to be loaded by the computing device, wherein at least controls of the media player are hidden from a user;
     - monitoring for interaction with the defined user interface element; and,
     - controlling the media player via the user interface element so as to cause the media player to function in a manner corresponding to the function associated with the user interface element.

2. The method of Claim 1, wherein the graphical user interface exists independent from any media player.

3. The method of Claim 1, wherein the graphical user interface comprises at least one graphics file.
4. The method of Claim 1, wherein the playlist further comprises a media type indicator associated with the first media file.

5. The method of Claim 4, wherein the media player capable of playing the first media file is determined, at least in part, by the media type indicator associated with the first media file.

6. The method of Claim 1, further comprising storing the graphical user interface and the association of the computer program process code with the graphical user interface in a manner interpretable by a web browser.

7. The method of Claim 6, wherein the graphical user interface, the association of the graphical user interface and the playlist, and the association of the graphical user interface and the computer program process code are stored in the same file.

8. A system for controlling a plurality of media players, comprising:

   a defined graphical user interface, wherein the defined graphical user interface receives user interactions, wherein the defined graphical user interface comprises a user interface element, wherein the defined user interface element has a function associated therewith, and wherein the defined graphical user interface is stored in a markup language derived from the Standardized Generalized Markup Language and interpretable by a web browser running on a computing device;

   a set of instructions, interpretable by the web browser, whereby the web browser can send commands to one of a plurality of media players installed on the computing device, thereby permitting the web browser to control playback of a media file based on user interactions with the defined graphical user interface; and

   a playlist, wherein the playlist is stored in a markup language derived from the Standardized Generalized Markup Language and interpretable by the web browser, wherein the playlist comprises a plurality of entries, and wherein at least a subset of the plurality of entries corresponds to a multimedia file.
9. The system of Claim 8, wherein the set of instructions further comprises instructions for determining whether a media player capable of playing a media file is installed on the computing device.

10. The system of Claim 9, wherein each playlist entry further comprises at least one media type indicator associated with the playlist entry.

11. The system of Claim 10, wherein the determination of whether a media player capable of playing the media file is installed on the computing device is based, at least in part, on the media type indicator associated with a playlist entry.

12. The system of Claim 8, wherein the playlist and the graphical user interface are stored using a common markup language.

13. The system of Claim 12, wherein the common markup language is the extensible Markup Language.

14. The system of Claim 11, wherein the common markup language is the Hypertext Markup Language.

15. A browser interpretable document comprising:

   a playlist comprising a plurality of entries, wherein each entry comprises a first media file or a pointer to the first media file;

   a graphical user interface definition, or a pointer to the graphical user interface definition, the graphical user interface definition comprising a first user interface element and a function associated with the first user interface element; and,

   a set of commands, or a pointer to the set of commands, for controlling the operation of a plurality of media players;

wherein, when the browser interpretable document is rendered by a conventional web browser, the web browser renders a graphical user interface based on the graphical user interface definition, and wherein the rendered graphical user interface controls one of the plurality of media players to facilitate playing of the first media file.
16. The browser interpretable document of Claim 15, wherein the graphical user interface definition is defined in a standardized markup language derived from the Standardized Generalized Markup Language.

17. The browser interpretable document of Claim 16, wherein the markup language is Hypertext Markup Language.

18. The browser interpretable document of Claim 17, wherein the set of commands comprises instructions in a scripting language.

19. The method of Claim 15, wherein each playlist entry further comprises a media type indicator associated with the first media file.

20. The method of Claim 19, wherein the set of commands comprises instructions through which the browser can determine whether a media player capable of playing the first media file is installed on a computing device on which the browser is running, and wherein such determination is made, at least in part, based on the media type indicator associated with the first media file.
Create a graphical user interface through which a plurality of media players are to be controlled.

Define at least one user interface element to be displayed as part of the graphical user interface.

Associate at least one function with the defined user interface element.

Associate at least one playlist with the graphical user interface, the playlist comprising a media file or a pointer to the media file.

Is a media player capable of playing the media file accessible to the computing device?

Yes: Cause computing device to load media player capable of playing the media file.

No: Obtain media player capable of playing the media file.

Hide media player controls from user.

Monitor user interaction with the user interface element(s).

Control the media player based on the user's interactions with the user interface element(s) so as to cause the media player to function in a manner corresponding to the function associated with the user interface element(s).

Figure 1
Mike's Music Blog

This is my music blog and playlist. There are many like it, but this one is mine.

1. [link]
   This always cracks me up. You know what? My pants are on fire. Just kidding.

2. [link]
   Bog and Dave -- this one goes out to you jarheads. Now give me back my wallet.

3. [link]
   A music video from copyleft record label Loco records -- TGF

4. [link]
   You better have heard this song in a while! Denise used to love it. It brings back old times in a big way.

5. [link]
   BAD BRAIN? YEAHHHHH... Can do these guys rock the house.

Marital status
single

Likes
Incredible music

Picture

Figure 2
YMU Playlist Demo

- 10,000 Years - Horovitz

- Got My Own Thing - Lil' Pooh

- And Your Bird Can Sing - Matthew Sweet & Susanna Hoffs

- Monster In Man - Jimi Castello

Figure 4
Figure 7
Figure 8

YMU Playlist Demo

400

410

420

430

440

700

810

820

830
YMU Playlist Demo

400

15,000 Years - Harem Anime

410

Got My Own Thing - Lil Phac

420

And Your Bird Can Sing - Matthew Skarr & You and Me

430

Monkey To Man - The Castles

700

810

820

Figure 9
YMU Playlist Demo

Figure 10
Figure 12
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>YMU Playlist Demo</title>
</head>
<body onselectstart="return false">
  <h1>YMU Playlist Demo</h1>
  <p>
    <a href="mms://mms.content.loudeye.com/600141/p/068/58/80/0685880_0108_00_0690.wma" class="htrack" title="10,000 Years" type="audio/ms-wma" tabindex="1"/>
    <img alt="10,000 Years" src="http://www.artistdirect.com/Images/Sources/AMGCOVERS/music/cover200/drg200/g223/g22312crhvi.jpg"/>
    10,000 Years - Honeydogs
  </p>
  <p>
    <a href="http://music.yahoo.com/track/24440930" class="htrack" title="Got My Own Thing" tabindex="2"/>
    Got My Own Thing - Liz Phair
  </p>
  <p>
    <a href="http://music.yahoo.com/track/31683488" class="htrack" title="And Your Bird Can Sing" tabindex="3"/>
    And Your Bird Can Sing - Matthew Sweet & Susanna Hoffs
  </p>
  <p>
    <a href="http://music.yahoo.com/track/16472906" class="htrack" title="Monkey to Man" tabindex="4"/>
    Monkey to Man - Elvis Costello
  </p>
  <script type="text/javascript" src="http://server.yahoo.com/user/file.js"></script>
</body>
</html>

Figure 13
And Your Bird Can Sing - Matthew Sweet

<a href="http://music.yahoo.com/track/24440930" title="Got My Own Thing" type="audio/ms-wma" tabindex="2">Somebody's Miracle</a>
Got My Own Thing - Liz Phair

<a href="http://music.yahoo.com/track/16472906" title="Monkey To Man" type="audio/ms-wma" tabindex="3">The Delivery Man</a>
Monkey To Man - Elvis Costello

<a href="http://music.yahoo.com/track/2127672" title="Breathe (2 AM) (Radio Version)" type="audio/ms-wma" tabindex="4">Breathe (2 AM)"</a>
Breathe (2 AM) (Radio Version) - Anna Nalick

<script type="text/javascript" src="http://server.yahoo.com/user/file.js"></script>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>My First Playlist</title>
</head>
<body>
<h1>My First Playlist</h1>
<p>
<a href="http://example.com/mp3" class="htrack" title="My Sharona"
tabindex="2" type="audio/mpeg">
<msg alt="The Knack album art"
src="http://example.com/img/albumart/the_knack.gif"/>
</a>
don't deny you love this song
</p>
<div class="htrack">
<img alt="hTrack logo" src="http://example.com/img/albumart/htrack.gif"/>
<a href="http://example.com/ogg" class="timed" tabindex="3">Hello, hTrack
World! [Ogg Vorbis] </a>
</div>
<a href="http://example.com/qt" class="timed" tabindex="4">Hello, hTrack
World! [Quicktime] </a>
<a href="http://example.com/about">Lyrics to this song</a>
<p>
<a href="http://example.com/intro" class="htrack" title="playlist intro"
tabindex="1">
<img alt="The Knack album art"
src="http://example.com/img/albumart/logo_for_my_show.gif"/>

this intro MP3 will always play first
</a>
</p>
</body>
</html>
A. **CLASSIFICATION OF SUBJECT MATTER**

**G06F 17/00(2006.01) I, G06F 19/00(2006.01) I**

According to International Patent Classification (IPC) or to both national classification and IPC

B. **FIELDS SEARCHED**

Minimum documentation searched (classification system followed by classification symbols)

IPC 8 : G06F 17/00; G09G 5/00; G06F 9/445

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Korean utility models and applications for utility models since 1975.

Japanese utility models and applications for utility models since 1975.

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

e-KIPASS(KIPO internal) "media player, graphical user interface, control"

C. **DOCUMENTS CONSIDERED TO BE RELEVANT**

<table>
<thead>
<tr>
<th>Category</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>US20040261040 A1 (MARK J. RADCLIFFE, et al.) 23 DECEMBER 2004 See the Abstract; Figure 1; Page 1</td>
<td>1-20</td>
</tr>
<tr>
<td>A</td>
<td>US20020161797 A1 (KEVIN T. GALLO, et al.) 31 OCTOBER 2002 See the Abstract; Figure 7; Pages 1-7</td>
<td>1-20</td>
</tr>
<tr>
<td>A</td>
<td>WO20086709 A2 (ARTECIA TECHNOLOGIES, INC.) 31 OCTOBER 2002 See the Abstract; Figures 1-16; Pages 1-7</td>
<td>1-20</td>
</tr>
<tr>
<td>A</td>
<td>US20030076345 A1 (CFIRIS FELLER, et al.) 24 APRIL 2003 See the Abstract; Figure 5; Pages 1-7</td>
<td>1-20</td>
</tr>
<tr>
<td>A</td>
<td>KR1020050072071 A (MICROSOFT CORP.) 08 JULY 2005 See the Abstract; Figure 5b; Pages 3-10</td>
<td>1-20</td>
</tr>
</tbody>
</table>

☐ Further documents are listed in the continuation of Box C. ☒ See patent family annex.

* Special categories of cited documents:
  * "A" document defining the general state of the art which is not considered to be of particular relevance
  * "E" earlier application or patent but published on or after the international filing date
  * "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of citation or other special reason (as specified)
  * "O" document referring to an oral disclosure, use, exhibition or other means
  * "P" document published prior to the international filing date but later than the priority date claimed
  * "T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
  * "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
  * "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
  * "&" document member of the same patent family

Date of the actual completion of the international search

07 AUGUST 2008 (07.08.2008)

Date of mailing of the international search report

07 AUGUST 2008 (07.08.2008)

Name and mailing address of the ISA/KR

Korean Intellectual Property Office
Government Complex-Deajeon, 139 Seonsa-ro, Seogu, Deajeon 302-701, Republic of Korea

Facsimile No. 82-42-472-7140

Authorized officer

Choi, Jae Gwi

Telephone No. 82-42-481-5787

Form PCT/ISA/210 (second sheet) (July 2008)
<table>
<thead>
<tr>
<th>Patent document cited in search report</th>
<th>Publication date</th>
<th>Patent family member(s)</th>
<th>Publication date</th>
</tr>
</thead>
<tbody>
<tr>
<td>US20020161797 A1</td>
<td>2002.10.31</td>
<td>NONE</td>
<td></td>
</tr>
<tr>
<td>KR1020050072071A</td>
<td>2005.07.08</td>
<td>CN1652964 A</td>
<td>2005.08.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP1557746 A2</td>
<td>2005.07.27</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JP17196784 A</td>
<td>2005.07.21</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US2005146534 A1</td>
<td>2005.07.07</td>
</tr>
</tbody>
</table>