Working Paper Series
ISSN 1170-487X

Melody based tune retrieval over the World Wide Web

**by David Bainbridge,
Rodger J McNab and Lloyd Smith**

Working Paper 98/17
November 1998

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

Melody based tune retrieval over the World Wide Web

David Bainbridge, Rodger J. McNab and Lloyd A. Smith

Department of Computer Science, University of Waikato, Hamilton, New Zealand.
{D.Bainbridge, R.McNab, L.Smith}@cs.waikato.ac.nz

Topic Categories

- Multimedia collections: access to a music archive
- Web-based interface.

Abstract. In this paper we describe the steps taken to develop a Web-based version of an existing stand-alone, single-user digital library application for melodical searching of a collection of music. For the three key components: input, searching, and output, we assess the suitability of various Web-based strategies that deal with the now distributed software architecture and explain the decisions we made. The resulting melody indexing service, known as MELDEX, has been in operation for one year, and the feed-back we have received has been favorable.

Key words: Music collection · Web-based audio · Distributed architecture · Human computer interaction design

1. Introduction

To develop a melody-based computer search facility for music, equivalent to the text-based searching facilities available for the written word, many difficult and challenging problems must be solved [MSBW97, MSWH96]. In the system developed by McNab *et al.*, a user sings a remembered fragment of song into a microphone, sets a few search options (such as whether to search for approximate or exact matches) and then directs the computer to search its database of tunes. To accomplish searching, the computer system must transcribe the audio signal into symbolic notes, then use this information to perform "string" matching on a text derived form of the music and musical database. Extending the availability and accessibility of this program through the World Wide Web raises a new set of interface problems due to the restrictive communication regime imposed by the medium.

In this paper we describe how an existing stand-alone search program written for the Macintosh computer was redeveloped as a Web-based service—similar in idea to Internet search engines such as AltaVista. In doing so we demonstrate how many of the restrictions imposed by human computer interaction design using HTML (Hyper Text Mark-up Language) can be overcome to provide what we believe to be a useful (and unique) Digital Library resource on the Web.

The structure of the paper is as follows: Section 2 describes the finished Web service and helps set the context of the problem; Section 3 describes how the audio input and search parameters are entered; Section 4 describes the modifications necessary to turn a graphically controlled, single-user, stand-alone search program into command-line controlled, multiple user server; Section 5 describes how we retain data between separate executions of the software; and Section 6 summaries our experiences with some

concluding remarks.

2. The final product

Figure 1: Default query page.

Figure 1 shows the default query page to the Web-based melody index service. First the filename of an acoustic sample is entered—this is the tune fragment that will be used to perform the search, and can be a digitized recording of the user singing or playing a tune on a musical instrument. Next the databases to search are selected. Currently we have four databases: North American/British, German, Chinese, and Irish folksongs derived from two sources: the Digital Tradition [Gre94] and the Essen collection [Sch92], totaling 9,400 melodies. The user can elect to search any number of the databases simultaneously. Alternatively, by clicking on the collection name it is possible to browse the tunes by title, sorted alphabetically.

With the digitized tune and databases specified, the final stage is to submit the query for processing. This is accomplished by pressing the "Quick search" button at the bottom of the page. In doing so, various search options not accessible from this particular query page are set to default values. These are:

Position:	only match the start of a tune;
Pitch:	classify sequences of notes as *up*, *down*, or *same* rather than exact intervals;

Rhythm:	ignore note durations;
Matching algorithm:	a fast, but crude, state matching algorithm; and
Tuning:	adapt to user's own tuning, rather than assume standard frequencies such as A-440.

When the search button is pressed, all this input information is packaged and sent to our server which parses the data, activates the appropriate search and sends the result back to the user. The result of processing Figure 1 is shown in Figure 2.

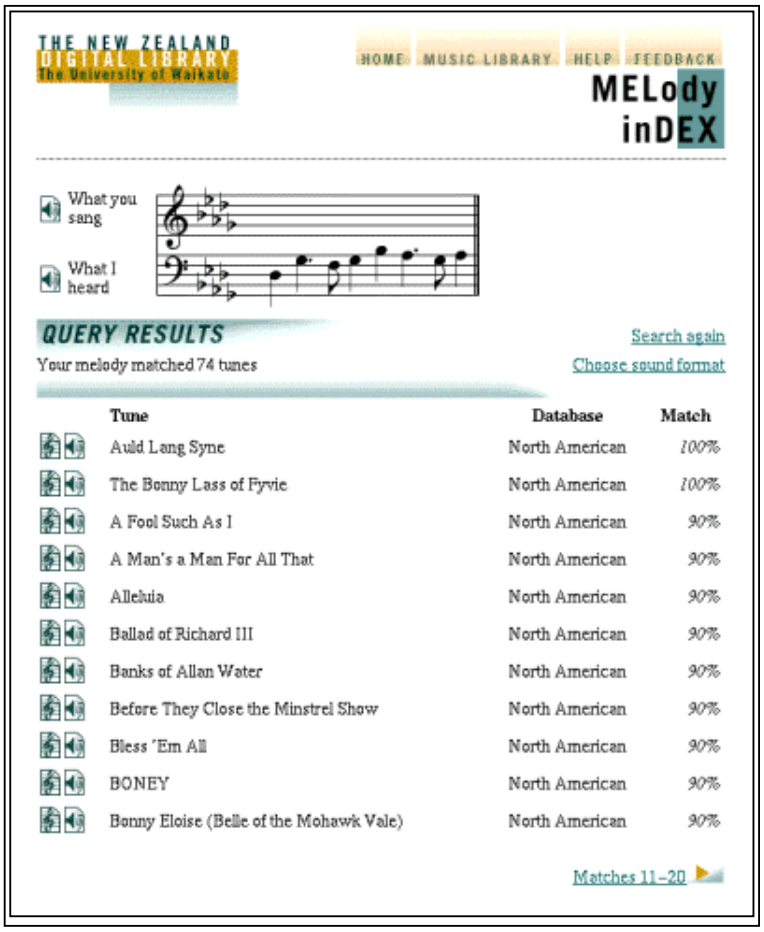

Figure 2: The result of the query shown in Figure 1.

As with other Internet search engines, our software ranks the returned database entries and formats them into pages with next and previous links. In this case 74 tunes were found, with *Auld Lang Syne* (the tune sung) the top item. By clicking on the "speaker" icon next to a song title the user will hear the piece played, assuming the browser has been configured properly; by clicking on the "treble clef" icon the user will be presented with a graphical version of the melody.

Below we detail the design decisions that lead us to this interface.

3. Collecting audio data

The physical task of transcribing a live audio signal is straightforward when the source (the user) and computer hardware are in the same room. The task becomes much harder when the microphone and audio processing hardware are thousands of miles apart. Although products such as Live RealAudio [Rea97]—which permits the live transmission of sports commentary, radio shows and the like over the Internet—prove it is technically feasible to do this, we did not wish to become tied to proprietary software (which can often limit the choice of computer platforms); nor was there, at the time of implementation, any standard support in HTML or its extensions for audio input.[1]

Our solution was to accept this lack of standardization and instead require users to step outside their Web browsers to record themselves singing/playing using what ever audio recording software is available on their computer.[2]

Allowing such a variety in recording programs does raise the issue of different audio formats: Sun's AU format, Window's WAV format and so forth. Such differences are dealt with using `sox`, a general utility sound exchange program written by Lance Norskog and others [Nor95]. Our core software only processes µlaw audio files; filter commands using `sox` transform other formats appropriately.

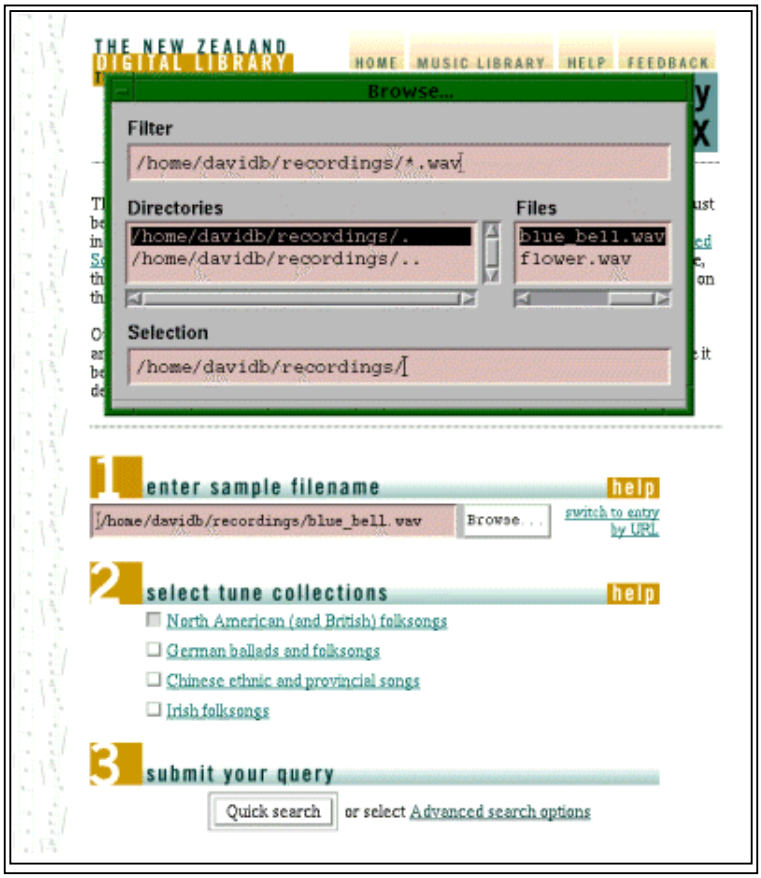

Figure 3: Default query page using file-upload.

The second issue in allowing users to record acoustic input off-line is the mechanism used by the music searching software to access this data. A solution that is guaranteed to work for any browser is to request the URL (Universal Resource Locator) for the acoustic file (Figure 1). A more elegant alternative is to

use the form type "File-upload" as is shown in Figure 3. Using this option a browse button appears to the right of the text area for Step 1 which, when pushed, causes a file browser popup window to appear. There the user can traverse their files to locate the desired recording. This eliminates the chance of typing mistakes that can happen when using the URL entry method.

Unfortunately, file-upload is currently a non-standard extension to HTML; however, many browsers support this (for instance Netscape 3.0 and above, and Internet Explorer 4.0). Thus, the final layer to file input is to use CGI (Common Gateway Interface) scripts written in Perl to interrogate a user's browser to determine what type and version number it is and consequently if file-upload is supported. This done, the CGI script generates the appropriate HTML page (Figure 1 or 3).

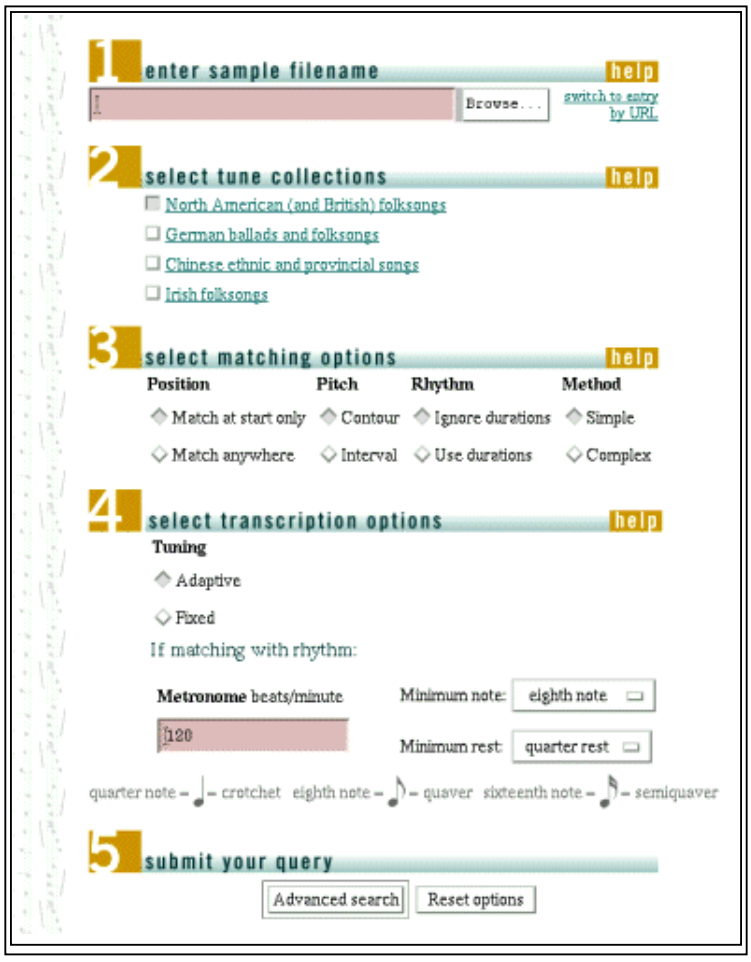

Figure 4: The advanced query page.

In addition to specifying the acoustic file, to perform a search the user must specify parameters that control the type of search performed. These input values—listed above—are more like text searching options such as case sensitive "on" and stemmed words "off", and can be dealt with in a similar way using standard HTML form attributes such as radio buttons and pull down menus. These options can be seen in Figure 4 which shows the advance query page.

4. Searching the database

In the original Macintosh application there is a close integration between the input phase and the search phase. This integration comes about naturally since the two phases are simply components of a larger software program. All input data is held in memory and passed as arguments to functions that perform the search. This direct arrangement is shown in Figure 5.

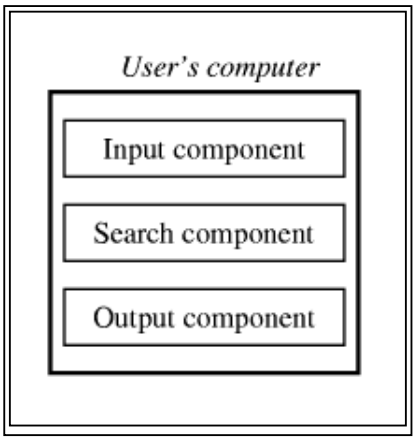

Figure 5: The relationship between input, search and output components in the stand-alone Macintosh program is straightforward.

In moving to a Web-based version, the first significant difference is the distributed nature of the environment that physically separates the logical phases. This is shown in Figure 6, where the computer handling data input may not even be on the same continent as the computer that performs the search!

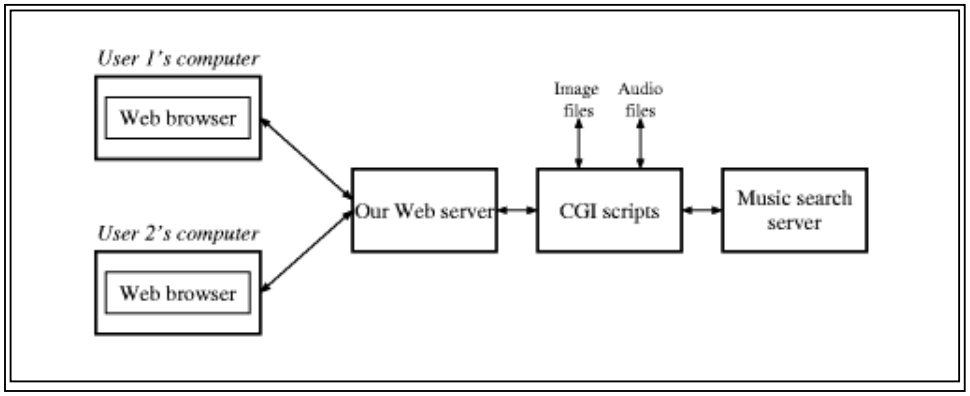

Figure 6: The relationship between input, search and output in the Web-based version is complex.

Transmitting input data over such a link is solved by using MIME (Multipurpose Internet Mail Extensions) types; the asynchronous interaction of the Macintosh based graphical user interface is replaced with a single submit operation once the data is gathered—all of this is provided by standard HTML. The first thing, however, that the processing software must do is to check that all the data fields specify valid values. In the event of an erroneous or missing data item, the CGI script generates a Web page detailing where the mistake has occurred.

Another key difference between the stand-alone and Web version is the move from a single user to multiple users. The solution here is to restructure the search software so it is harnessed inside a server

program (rather than respond to input requests directly). This is accomplished using a text-based message passing scheme implemented using Unix sockets [WS91].

Input takes the form of single line requests where the first word defines the requested operation and subsequent arguments appear where appropriate. Output (described more fully below) often requires more than one line to transmit all its information; consequently, the token "END" is reserved as a command word and used as a delimiter to mark the end of such a sequence. It is not necessary to clarify the type of output being produced since the program generating the input request will know what type of output data to expect.

5. Displaying the results

In the original Macintosh application there is once again close integration between search and displaying the results. Lists of matches are held in memory and used to determine what is shown to the user, where control of the graphics display includes the color of individual pixels. With the move to a Web environment, the ability to retain data and track state information becomes harder. And while it is true that HTML is a graphical medium, in its rudimentary form there is no interactive coloring of pixels.

The requirement to graphically display pages of music could be met by using a plug-in product such as NoteView [Jac97] which allows music to be encoded and sent over the Web at a symbolic level (more compact than a graphical representation) but still have it displayed graphically by the browser. However, this would once again tie us to a proprietary product which we want to avoid if possible. Instead we chose to generate GIF (Graphics Interchange Format) images of music that are then incorporated into HTML documents.

To retain data in a CGI script, the two main mechanisms are intermediate files and cookies. A cookie is a bit like an associative array where, if you know the identification key specified at its creation time, you can access the data elements stored in the array through its text index values. Cookies are an attractive solution to managing the list of matching tunes returned from a user’s search. Using a cookie, the list could be stored as a dynamic array of tuple values (tune name, database collection name, percentage match) which are then retrieved later to generate a specific page of matches (for example, matched items 11-20).

In actual fact, a cookie is implemented as a file. An important difference, however, is that files generated by a CGI script are on the server side, but the file used in a cookie is on the client (the user) side. This means it is easy for a CGI script to distinguish between requests made by different users, however it is this very difference that makes cookies a potential security threat to a user and therefore provided as a browser option that can be switched off.

Fortunately, it is not necessary to resort to cookies to store a list of matched tunes—CGI scripts using server-side files are more than adequate. When a new search is started, a unique file name is generated, in which the results of the search are stored, and a web page with up to the first 10 matches generated. Subsequent requests for next or previous pages are encoded as URL with the appropriate CGI arguments embedded in it, for example:

```
http://www/cgi-bin/meldex/hitpages?lower_index=10&filename=27674_882226100.data
```

After a specified amount of time the data file is deleted.

Storing user preferences (state information) on the server side is more problematic. This requires the generation of a unique filename based on the user's name—a tricky task when not all browsers set this information and certain computer configurations do not even require the user to log in before accessing the Web. In this situation cookies are best, and we use cookies to store a user's preferred audio type for playback.

The first time a user clicks on the speaker icon (Figure 2) for playback, a new page is generated that offers a variety of audio formats. Once the user has selected one of these, that choice is stored in a cookie and the requested tune is played. Subsequent clicks on a speaker icon will automatically use this file type to play the melody. Of course just because a user has chosen a particular audio file type does not mean they always want this, and options exist to revisit the "audio file type" page.

If a user chooses to have cookies disabled in their browser, the worst that happens is that each time they click on a speaker icon to hear a piece, they will have to pass through the intermediate page where they select the type of audio file they want.

6. Conclusions

In this paper we have described how a single-user, stand-alone melody based tune retrieval system was adapted to work over the Web. While it is near impossible to provide a Web service that is identical to the original product, it is possible to develop an interface which—in the context of Web interfaces—reduces the cognitive load placed on the user. Examples of this include the use of the HTML form type "file-upload" to select an acoustic recording and cookies to record the user's preferred audio file type for playback.

The resultant user interface is similar in appearance to Internet search engines, and significant effort has been spent in program design to ensure this comparability exists. In fact our architecture design conforms to a more general structure designed for the New Zealand Digital Library project (`http://www.nzdl.org/`) that accommodates text, music and other multi-media collections such as oral history [MWB97].

In reaching as broad a range of users as possible, our experience is that software should avoid using platform specific plug-ins and proprietary software, relying on only the lowest common denominator. In situations where a non-standard extension is used to improve the interface, the software's functionality must not be compromised if the extension is missing for a particular user. Our use of file-upload and cookies meet this criteria. Hopefully standardization will improve in the future, allowing more dynamic applications.

At the time of writing (December '97) the melody index service has been in operation for one year and all the feedback we have received has been positive.

MELDEX can be accessed via the URL "http://www.nzdl.org/meldex".

Footnotes

[1] This will change in time. For instance, the latest version of Java Media Framework API (currently in beta form) carries support for both recording and playback of sound.

[2] We supply a help page with advice on software products to permit audio recordings and links to freely available programs for common computer platforms.

References

[Gre94]	Greenhaus, D. (1994) About the Digital Tradition. "http://www.deltablues.com/".
[Jac97]	Jacobson, M.N. (1997) *Nightingale Version 3: Integrated notation software for the Macintosh computer,* Jazz Player, No. 2, pp. 29-31.
[Nor95]	Norskog, L. (1995) SoX: Sound eXchange Home Page "http://www.spies.com/Sox/".
[NSBW97]	McNab, R.J., L.A. Smith, D. Bainbridge and I.H. Witten (1997) *The New Zealand Digital Library MELody inDEX* , D-lib Magazine, "http://www.dlib.org/dlib/may97/meldex/05witten.html".
[MSWH96]	McNab, R.J., L.A. Smith, I.H. Witten and C. Henderson (March 1996) *Towards the digital music library: tune retrieval from acoustic input,* Proc. ACM Digital Libraries Conference, Bethesda, Maryland, pp. 11-18.
[NWB97]	McNab, R.J., I.H. Witten and S.J. Boddie (1997) *A distributed digital library architecture incorporating different index styles,* Department of Computer Science, University of Waikato, Hamilton, New Zealand (in preparation).
[Rea97]	RealNetworks Home Page (1997) "http://www.real.com/".
[Sch92]	Schaffrath, H. (1992) *The ESAC databases and MAPPET software,* Computing in Musicology, Vol 8., W. Hewlett and E. Selfridge-Field (eds), Menlo Park, California: Center for Computer Assisted Research in the Humanities.
[WS91]	Wall, L. and R.L. Schwartz (1991) *Programming Perl*, O'Reilly & Associates.

Dr. David Bainbridge, Lecturer
Department of Computer Science, University of Waikato, Hamilton, New Zealand
Email: davidb@cs.waikato.ac.nz

Last modified: Tue Dec 15 16:25:59 NZDT 1997