Querying CDDB in Lazarus

Michaël Van Canneyt

October 5, 2008

Abstract

FreeCDDB is a database with tracks for audio CD. This database is queried by many popular audio programs that play audio CDs. The CDDB format and protocol are quite simple. This article shows how to query the CDDB

1 Introduction

Most applications capable of playing an Audio CD or tools to rip audio CDs or even CD-cover design programs have the ability to query an online database with audio-cd definitions. This enables them to show the titles and performer of the CD's tracks. FreeCDDB.org is one of the providers of this online service: it uses a simple database format and an easy-to-use protocol. Free Pascal or Lazarus users can easily query this database with the standard tools provided by Lazarus and FPC.

The CDDB database has a file for each known CD. The file format used by FreeCDDB is quite simple, as it is text based. Each database entry consists of a set of `Key=Value` pairs, where each pair is on a line. Each audio disc entry is identified by a DISCID key, followed by an 8-digit hexadecimal digit. After this key, the properties of the disc follow. Basic entries are

DTITLE The disk's performer and title, separated by a slash.

DEXT Extra disk information (a comment). Multiple entries can exist.

TTITLEN The title of track N. N starts at zero.

EXTTN Extra information for track N.

PLAYORDER A playing order for the tracks.

There are several FreeCDDB protocols: There is a protocol over plain TCP/IP, this is the native protocol. The Protocol is text based and quite simple. The client sends a CDDB command, and the server replies with a status line, possibly followed by the content of the response. The status line follows the usual TCP/IP text protocol conventions: it starts with a numerical response code, followed with a more elaborate response string.

This protocol has been put on a HTTP transport layer: the client executes a HTTP GET or Post command, with specially formatted CDDB command, and the server sends the response as a HTTP document back to the client. The response is identical to the one that would be sent with the plain TCP/IP transport.

For instance, the following represents a query command and it's response:

```
cddb query b70e160e 14 150 19552 42350 75757 103260 117062\
131722 147605 163365 188055 203282 218230 235580 257545 3608
200 data b70e160e Buena Vista Social Club/Buena Vista Social Club
```

2 CDDB commands

There are 2 important commands in the CDDB protocol:

query Checks for the presence of an entry in the CDDB database for an audio disk. The disc is identified by a series of numbers, which essentially represent the duration of the tracks, preceded by a hash value, the DiscID. The server will respond by sending series of disc entries that match the query. For each entry, the DiscID, category and title/performer is sent.

read Reads the entry corresponding to a certain DiscID/Category. The response (if any) is the complete file for that disc.

The response of both commands must be used to be able to show the contents of a disc.

The CDDB file can be read with the `TCDDBParser` component in the fpCDDB unit. This component introduces a couple of classes to describe the response:

```
TCDTrack = Class(TCollectionItem)
  Property Title : String;
  Property Performer : String;
  Property Extra : String;
  Property Duration : TDateTime;
end;

TCDTracks = Class(TCollection)
  Property Track[AIndex : Integer] : TCDTrack;
end;

TCDDisk = Class(TCollectionItem)
  Property PlayOrder : String;
  Property Year : Word;
  Property Title : String;
  Property Performer : String;
  Property Extra : String;
  Property DiscID : String;
  property Tracks : TCDTracks;
end;

TCDDisks = Class(TCollection)
  Property Disk[AIndex : Integer] : TCDDisk;
end;
```

The various properties are simple and self-explaining. The `TCDDBParser` component has a `Disks` property of type `TCDDisks`, which can be populated with the method

```
Function ParseCDDBReadResponse(Response: TStream;
                               WithHeader: Boolean = True): Integer;
```

This function takes as input the response from the CDDB 'read' command. If `WithHeader` is `True`, then the first line will be treated as the command status. It will be examined and if the status is OK then the rest of the lines will be parsed. The function returns the number of disk entries it found in the response, and the `Disks` property will be filled with the definitions found in the response.

The CDDB 'query' command returns one or more disc definitions that match the query parameters. The response is described by the following classes in the fpcddb unit:

```
TCDDBQueryMatch = Class(TCollectionItem)
  Property DiscID : Integer;
  Property Category : String;
  Property Title : String;
  Property Performer : String;
end;

TCDDBQueryMatches = Class(TCollection)
  Property Match[AIndex : Integer] : TCDDBQueryMatch;
end;
```

The `TCDDBParser` component offers the following method to examine the result of the 'query' command:

```
Function ParseCDDBQueryResponse(Response : TStream;
                        Matches : TCDDBQueryMatches;
                        WithHeader : Boolean = True) : Integer;
```

It is similar in structure to the `ParseCDDBReadResponse` command. On return, the matches collection is filled with the matches reported by the CDDB server. The CDDB server can report an exact (1 disc entry) or a fuzzy match (more than 1 entry). It does this with different command result statuses (200 and 210). The method can handle both cases.

These 2 calls serve to interpret the response of the CDDB server: the result is a collection of disc and track information.

3 Modus operandi

To compose this information from a CDDB server is a 4 step process:

1. Scan the audio CD. This results in a list of tracks, with their sizes.
2. Compose a Disc ID from the result of the previous step.
3. Execute a CDDB query command using the disc ID. This will result in a set of matches.
4. Use the DiscID and Category from one of the matches to read the disc entry.

Each of these steps can be done with Lazarus/FPC:

- Scanning the audio CD can be done with the cdrom unit. This unit works on Windows and Linux, and should be easily extendable to other FPC supported platforms.
- Calculating the DiscID can be done with the discid unit. It calculates the result based on the structures returned by the cdrom unit.
- Querying the CDDB server using HTTP can be done with any of the TCP/IP stacks that supports the HTTP protocol. The example uses synapse, because it is simple to use.
- Interpreting the CDDB commands can be done with the `TCDDBParser` component.

These steps will be demonstrated using a sample application. The application can scan an audio CD, query a CDDB server and display the result. It can also maintain and reuse a local cache of previous results, or can simply open and show a CDDB file.

The source of the program can be found on the disc accompagnying this issue. The `TMainForm` is the main form of the program, and this is where the bulk of the work is done. The `QueryCD` method is the most important one, it shows the 4 steps outlined above:

```
procedure TMainForm.QueryCD(Const ADevice : String);

Var
  TheDiscID,Tracks, i : Integer;
  Entries   : Array[1..100] of TTocEntry;
  Q,URL,Category     : String;
  M : TMemoryStream;

begin
  Tracks := ReadCDTOC(ADevice,Entries);
  If (Tracks<=0) then
    begin
    ShowMessage(Format(SErrFailedToReadCD,[ADevice]));
    Exit;
    end;
  TheDiscID:=CDDBDiscID(Entries,Tracks);
  If FUseCache and LoadFromCache(TheDiscID) then Exit;
  Q:=GetCDDBQueryString(Entries,Tracks);
  Category:=GetCategory(Q);
  If (Category='') then exit;
  M:=TMemoryStream.Create;
  try
    If GetDiskContent(DiscIDToStr(TheDiscID),Category,M) then
      begin
      ShowCDDB(M);
      If FUseCache then
        SaveToCache(DiscIDToStr(TheDiscID),M);
      end;
  finally
    M.Free;
  end;
end;
```

The `ReadCDTOC` call from the cdrom unit reads the contents of a disc, and returns the number of tracks. The track information is stored in the `Entries` array. This array is then used to calculate the `DiscID` number using the `CDDBDiscID` method from the discid unit. If the cache should be used and the file exists in the cache, it is read and the routine exits.

Otherwise, a CDDB query string is computed and is sent to the CDDB server in the `GetCategory` call. It returns the category of the disc, which can then be used to retrieve the entry from the `CDDB` server using the `GetDiskContent` call. If the call was succesful, the result is shown and saved to the file cache if the cache is in use.

The `GetCategory` call constructs a CDDB command: this is a simple command string, formatted with the `AQuery` sequence. The command is executed with the `DoCDDBCmd` function, which returns the server's response in the stream `S`. This result is examined using the `TCDDBParser` component:

```
Function TMainForm.GetCategory(AQuery : String) : String;
```

```
Const
  SCmdQuery = 'cmd=cddb+query+%s';

Var
  S : TMemoryStream;
  M : TCDDBQueryMatches;
  I : integer;

begin
  Result:='';
  M:=TCDDBQueryMatches.Create(TCDDBQueryMatch);
  try
    S:=TMemoryStream.Create;
    try
      If not DoCDDBCmd(Format(SCMdQuery,[AQuery]),S) then
        Exit;
      I:=FCDDB.ParseCDDBQueryResponse(S,M,True);
    finally
      S.Free;
    end;
    I:=SelectMatch(M);
    If I<>-1 then
      Result:=M[i].Category;
  finally
    M.Free;
  end;
end;
```

The `SelectMatch` method checks the number of matches. If there is one match, it is returned. If there are multiple matches, it pops up a dialog that allows the user to select a match. The category of the selected match is the result of the function.

The obtained (or selected) category is then used in the `GetDiskContent` function to get an entry from the CDDB server:

```
Function TMainForm.GetDiskContent(ADiscID,ACategory : String;
                                  Content : TStream) : Boolean;

Const
  SCmdRead  = 'cmd=cddb+read+%s+%s';

begin
  Result:=DoCDDBCmd(Format(SCmdRead,[ACategory,ADiscID]),Content);
end;
```

Executing a CDDB command is donoe using the `THTTPSend` class from `synapse`:

```
Function TMainForm.DoCDDBCmd(CMD : String;
                             Response : TStream) : Boolean;

Var
  Http : THTTPSend;
  U,URL : String;
```

```
begin
  Result:=False;
  U:=Fuser;
  If U='' then U:='Anonymous';
  HTTP:=THTTPSend.Create;
  Try
    Url:=FHostURL+'?';
    URL:=URL+StringReplace(Cmd,' ','+',[rfReplaceAll]);
    URL:=URL+'&'+Format(SHello,[U,FAppName,Version]);
    URL:=URL+'&proto=1';
    Result:=HTTP.HTTPMethod('GET',URL);
    If Not Result then
      ShowMessage(SErrQueryFailed)
    else
      begin
      Response.CopyFrom(HTTP.Document,0);
      Response.Position:=0;
      end;
  Finally
    HTTP.Free;
  end;
end;
```

The URL to fetch is puzzled together from various pieces:

- The base URL, which is the cddb.cgi CGI program on the cddb server. The `FHostURL` can be configured by the user, so a mirror can be configured.
- The `cmd` argument, containing the actual command.
- A `hello` login argument. It needs a username, application name and version number.
- Finally, a `proto` argument, which gives the protocol version number.

All this is put together and fed to the `HTTPSend` class. If the command response was succesfully received from the HTTP server, the response is copied to the response class.

Finally, the `ShowCDDB` method parses the response from the server, and if it is parsed succesfully, the first disk in the `Disks` collection is shown.

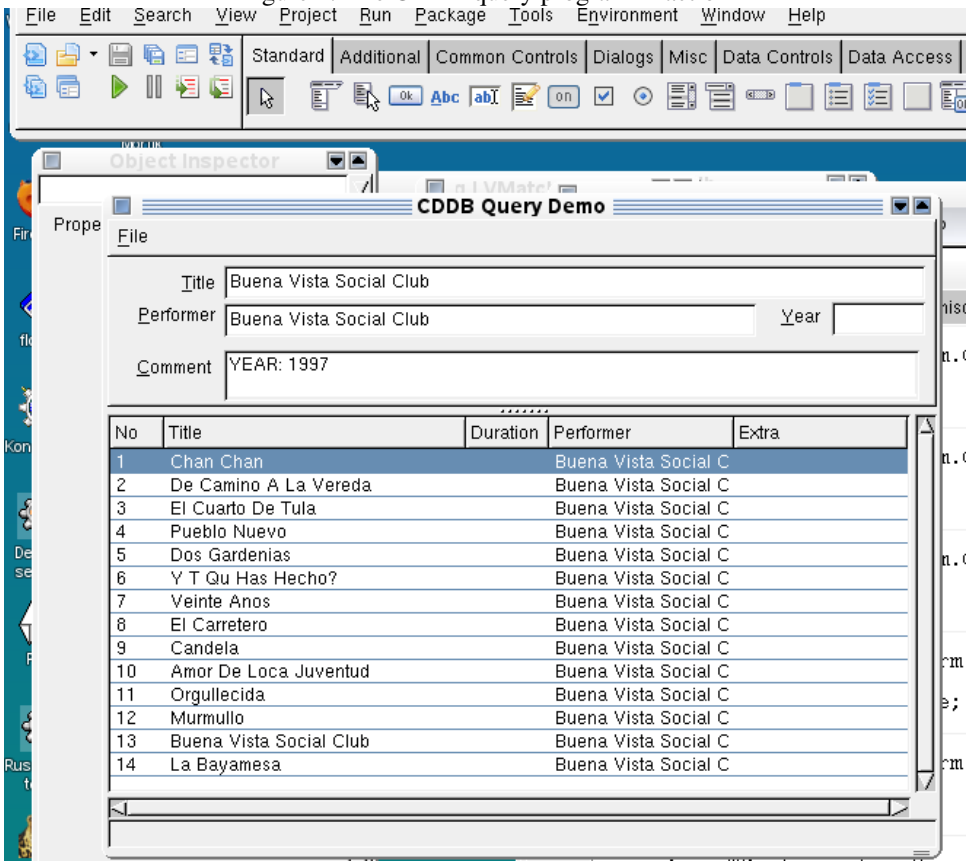
```
Function TMainForm.ShowCDDB(AStream : TStream;
                            WithHeader: Boolean = True): Integer;

begin
  Result:=FCDDB.ParseCDDBReadResponse(AStream,WithHeader);
  If (Result>=1) then
    ShowDisk(FCDDB.Disks[0]);
end;
```

The showdisk is a simple copying of the various properties of the disk and tracks properties to the various controls in the form. This code is not very instructive and will not be shown here. The final result can be seen in figure 1 on page 7.

The program also contains a configuration screen, where the CD reader device must be selected, the URL for the CDDB server can be configured, and the cache directory can be

Figure 1: The CDDB query program in action



set. Also the username used in the CDDB `hello` command can be specified there. The only interesting piece of code is in the `OnCreate` event, where the CD-DRive combobox is filled with the possible CD reader devices:

```
procedure TConfigurationForm.FormCreate(Sender: TObject);

Var
  Drives : Array[1..10] of String;
  I,Count : Integer;

begin
  Count:=GetCDRomDevices(Drives);
  For I:=1 to Count do
    CBCDRom.Items.add(Drives[i]);
end;
```

The `GetCDRomDevices` call from the cdrom unit returns the number of CD reader devices found, and returns the device strings (driveletters under Windows, device filenames under linux) in the `Drives` argument.

4 Conclusion

With all the routines presented here, the basic routines for querying a CDDB server have been explained. even though it is far from finished - e.g. the duration is not shown - The example program shows all that is needed to get started with CDDB in FPC/Lazarus: the basic units are all present in FPC/lazarus. The fpcddb unit is not present in the 2.2.2 release of FPC (only in Subversion or snapshots), so it is provided with the sources of the sample program.