DISCID Howto

Here is the algorithm, that generates a valid disc ID.
This text is from the freedb howto, which is based on the cddb howto by Ti Kan and Steve Scherf. You can download the freedb howto from the Download/Misc section.

If you are interested in a Perl-Version, check out jwz's make-freedb.pl script
You can also find Windows-programs (and the fully commented C-sourcecode for them) and the C-sourcecode for a Linux-program, that calculates the disc ID under Download/Misc on this site.

```
CDDB/FREEDB DISCID
------------------

Both forms (local and remote) of freedb access requires that the software
computes a "disc ID" which is an identifier that is used to access the
freedb.  The disc ID is a 8-digit hexadecimal (base-16) number, computed
using data from a CD's Table-of-Contents (TOC) in MSF (Minute Second Frame)
form.  The algorithm is listed below in Appendix A.

It is crucial that your software computes the disc ID correctly. If it does not
generate the disc ID correctly, it will not be compatible with the freedb.
Moreover, if your software submits freedb entries with bad disc IDs to the
freedb archives, it could compromise the integrity of the freedb.

We suggest installing one of the disc ID generator programs listed on the
freedb web page in the downloads/misc section, and then testing the disc
ID code in your software by comparing the disc ID generated by the program
with that of your software for as large a number of CDs as possible.
Alternatively you can e.g. use xmcd and compare the DiscID generated by xmcd
with that of your software. Bugs in disc ID calculation can be subtle, and
history shows that it sometimes takes hundreds of discs to find problems.

[...]

APPENDIX A - CDDB/FREEDB DISCID ALGORITHM
-----------------------------------------

The following is a C code example that illustrates how to generate the
CDDB/freedb disc ID. [...] A text description
of the algorithm follows, which should contain the necessary information
to code the algorithm in any programming language.


struct toc {
        int     min;
        int     sec;
        int     frame;
};

struct toc cdtoc[100];

int
read_cdtoc_from_drive(void)
{
        /* Do whatever is appropriate to read the TOC of the CD
         * into the cdtoc[] structure array.
         */
        return (tot_trks);
}

int
cddb_sum(int n)
{
        int     ret;

        /* For backward compatibility this algorithm must not change */

        ret = 0;

        while (n > 0) {
                ret = ret + (n % 10);
                n = n / 10;
        }

        return (ret);
}

unsigned long
cddb_discid(int tot_trks)
{
        int     i,
                t = 0,
                n = 0;

        /* For backward compatibility this algorithm must not change */

        i = 0;

        while (i < tot_trks) {
                n = n + cddb_sum((cdtoc[i].min * 60) + cdtoc[i].sec);
                i++;
        }

        t = ((cdtoc[tot_trks].min * 60) + cdtoc[tot_trks].sec) -
            ((cdtoc[0].min * 60) + cdtoc[0].sec);

        return ((n % 0xff) << 24 | t << 8 | tot_trks);
}

main()
{
        int tot_trks;

        tot_trks = read_cdtoc_from_drive();
        printf("The discid is %08x", cddb_discid(tot_trks));
}
```

This code assumes that your compiler and architecture support 32-bit integers.

The cddb_discid function computes the discid based on the CD's TOC data in MSF form. The frames are ignored for this purpose. The function is passed a parameter of tot_trks (which is the total number of tracks on the CD), and returns the discid integer number.

It is assumed that cdtoc[] is an array of data structures (records) containing the fields min, sec and frame, which are the minute, second and frame offsets (the starting location) of each track. This information is read from the TOC of the CD. There are actually tot_trks + 1 "active" elements in the array, the last one being the offset of the lead-out (also known as track 0xAA).

The function loops through each track in the TOC, and for each track it takes the (M * 60) + S (total offset in seconds) of the track and feeds it to cddb_sum() function, which simply adds the value of each digit in the decimal string representation of the number. A running sum of this result for each track is kept in the variable n.

At the end of the loop:
1. t is calculated by subtracting the (M * 60) + S offset of the lead-out minus the (M * 60) + S offset of first track (yielding the length of the disc in seconds).

2. The result of (n modulo FFh) is left-shifted by 24 bits.

3. t is left shifted by 8.

The bitwise-OR operation of result 2., 3. and the tot_trks number is used as the discid.

The discid is represented in hexadecimal form for the purpose of xmcd cddb file names and the DISCID= field in the xmcd cddb file itself. If the hexadecimal string is less than 8 characters long, it is zero-padded to 8 characters (i.e., 3a8f07 becomes 003a8f07). All alpha characters in the string should be in lower case, where applicable.

Important note for clients using the MS-Windows MCI interface:

The Windows MCI interface does not recognize data tracks, as you find them on CD Extra CD's. Therefore a wrong disc ID is generated for CD Extra's when using the MCI interface to read the TOC. Because of this, using the MCI interface should only be the last resort - if possible you should use other methods to read the TOC, like ASPI calls. An example disc ID calculator using ASPI can be found on the freedb website along with the sourcecode.
If for some reason, there is no other way for your program, than to use the MCI interface, here is the description how to do so:
The Windows MCI interface does not provide the MSF location of the lead-out. Thus, you must compute the lead-out location by taking the starting position of the last track and add the length of the last track to it. However, the MCI interface returns the length of the last track as ONE FRAME SHORT of the actual length found in the CD's TOC. In most cases this does not affect the disc ID generated, because we truncate the frame count when computing the disc ID anyway. However, if the lead-out track has an actual a frame count of 0, the computed quantity (based on the MSF data returned from the MCI interface) would result in the seconds being one short and the frame count be 74. For example, a CD with the last track at an offset of 48m 32s 12f and having a track length of 2m 50s 63f has a lead-out offset of 51m 23s 0f long. Windows MCI incorrectly reports the length as 2m 50s 62f, which would yield a lead-out offset of 51m 22s 74f, which causes the resulting truncated disc length to be off by one second. This will cause an incorrect disc ID to be generated. You should thus add one frame to the length of the last track when computing the location of the lead-out.

The easiest way for Windows clients to compute the lead-out given information in MSF format is like this:

```
(offset_minutes * 60 * 75) + (offset_seconds * 75) + offset_frames +
(length_minutes * 60 * 75) + (length_seconds * 75) + length_frames + 1 = X
```

Where X is the offset of the lead-out in frames. To find the lead-out in seconds, simply divide by 75 and discard the remainder.