Revolution

Optical Disc Drive Library (DVD)

Version 1.00

The contents in this document are highly confidential and should be handled accordingly.

RVL-06-0041-001-B
Released: October 22, 2007

Contents

Code Examples

Equations

Figures

Tables

Revision History

Version	Date Revised	Item	Description	Revised By
1.00	2006/03/01	-	First release by Nintendo of America Inc.	-

1 Overview

Wii uses an optical disc drive for game media. Since this device is based on DVD technology, function names in the optical disc drive library have “DVD” as a prefix.

This document covers the following topics:

- "2 Main Features" on page 7 introduces you to the main features of the Wii optical disc drive. If you are not a programmer, you can get basic information on the optical disc drive by reading this chapter. If you are a programmer, this chapter is a good starting point for you.
- "3 Simple Demo" on page 8 is a tour of the Wii optical disc drive through the use of a simple code demo.
- "4 How to Use APIs" on page 10 describes the APIs and how to use them.
- "5 Error Handling" on page 16 describes APIs for error handling.
- "6 File System" on page 18 briefly describes the Wii file system.
- "7 Directory Access Support Functions" on page 20 describes how to use directory access support functions.
- "8 Optical Disc Drive FAQs" on page 23 offers a FAQ list.

2 Main Features

Here are the main features of the Wii optical disc drive:

- Huge storage size
- Fast read speed
- Simple file system
- Easy error handling

2.1 Size

The total data storage capacity of a Wii disc is 4,699,979,776 bytes. However, about 140 megabytes are reserved as the system region.

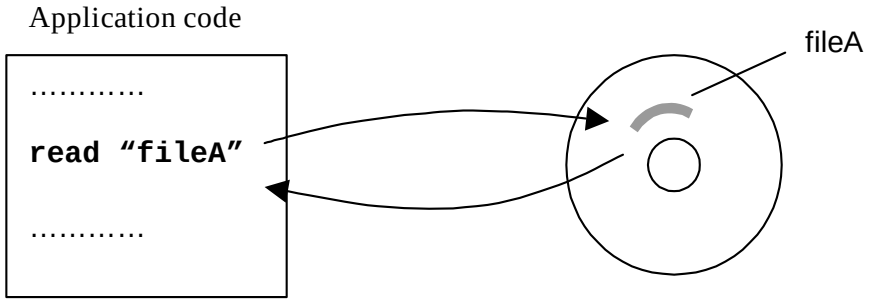
2.2 Read Speed

The read speed for a Wii disc is equivalent to the maximum DVD 6x. The Wii disc is always rotating uniformly, and data is recorded with the same density throughout the whole disc.

2.3 File System

The Wii disc can be accessed by symbolic string names, so you can read files by name. You don't have to specify the position of the file.

Figure 2–1 Applications Use Filenames to Retrieve Data From the Wii Disc



2.4 Error Handling

APIs automatically handle common user errors such as "no disc in drive," and so on, so you don't have to write complicated error-handling programs.

3 Simple Demo

Here is some simple code to read a file named `testfile1.txt` from the optical disc into a prepared buffer. This demo is stored as `dvddemo/src/dvddemo1.c`. Please compile it and see how it works.

Code 3–1 Read File Demo

```
1:      // The name of the file we are going to read.
2:      char*   fileName = "testfile1.txt";
3:
4:
5:      void main(void)
6:      {
7:          DVDFileInfo fileInfo;
8:          u32         fileSize;
9:          u8*         buffer;         // pointer to the buffer
10:
11:         // Init the OS and heap
12:         MyOSInit();
13:
14:         DVDInit();
15:
16:         // We MUST open the file before accessing the file
17:         if (FALSE == DVDOpen(fileName, &fileInfo))
18:         {
19:             OSHalt("Cannot open file");
20:         }
21:
22:         // Get the size of the file
23:         fileSize = DVDGetLength(&fileInfo);
24:
25:         // Allocate a buffer to read the file.
26:         // NOTE: Pointers returned by OSAlloc are always 32byte aligned.
27:         buffer = (u8*)OSAlloc(OSRoundUp32B(fileSize));
28:
29:         // read the entire file here
30:         if (0 > DVDRead(&fileInfo, (void*)buffer, OSRoundUp32B(fileSize), 0))
31:         {
32:             OSHalt("Error occurred when reading file");
33:         }
34:
35:         // From here, we can use the file
36:         :
37:         // Close the file
38:         DVDClose(&fileInfo);
39:         :
40:         OSFree(buffer);
41:
42:         OSHalt("End of demo");
43:
44:         // NOT REACHED HERE
45:     }
```

The lines of code are referenced by number in the following explanations:

7: At this line, you declare a structure called `DVDFileInfo`. This structure holds the information for the target file, such as the start position and the length. You don't have to fill the structure yourself; `DVDOpen` handles this (see line 17).

12: Call a local function, `MyOSInit()` (not shown here), to initialize the OS and heap. The heap will be required to use `OSAlloc()` later. Refer to the Operating System Guide in for the explanation of how to use `OSAlloc()`.

14: You must call `DVDInit()` before you issue any function related to the optical disc. This function handles optical disc access initialization, such as setting the interrupt handler, and so on.

17: ***Open*** the file here. `DVDOpen()` converts a filename into its file information. When debugging, we recommend checking the return value to know whether the open is successful.

23: `DVDGetLength()` is a macro to get the size of the file, which is necessary so that this code knows how large a buffer to allocate for the file to be read.

27: Allocate a buffer to read the file.

Note: We have to round up the file size to a multiple of 32 due to drive hardware restrictions. The size of a read *must* be a multiple of 32.

30: This is the way to read files. With `DVDRead()`, you can read a file synchronously, which means that the function returns when the read finishes. Since this function returns –1 if the read fails, you should check the return value. (For explanations of the arguments, as well as details on both synchronous and asynchronous reads, see "4.3 Read" on page 10.)

38: If you no longer need to access the file, you should ***close*** it.

Notes:

- There is no need to invalidate the data cache; `DVDRead()` does that by default.
- In your real games, you should include code for error checking. This is simple to do; see "5 Error Handling" on page 16 for more details.

4 How to Use APIs

4.1 Initialization

Code 4–1 DVDInit()

```
#include<revolution/dvd.h>

void DVDInit(void);
```

You must call this function before you can access the optical disc drive. `DVDInit()` handles all necessary initialization of the drive. You don't need to call this function more than once, but it doesn't cause any trouble if you do.

4.2 Open

Code 4–2 DVDOpen()

```
#include<revolution/dvd.h>

BOOL DVDOpen(char* fileName, DVDFileInfo* fileInfo);
```

In order to access a file, you first have to get the `DVDFileInfo` structure of the file by calling `DVDOpen()`. The information on the file *fileName* is retrieved and stored to *fileInfo*.

Filenames can take one of the following styles:

- `/texture/mario`
- `texture/mario`

If you use the former style, the optical disc drive device driver takes it as an absolute path name; in contrast, if you use the latter style, it's taken as a relative path name to the current directory by the device driver. There's no restriction on the length of the filename.

Like other file systems, you can use "`..`" (parent directory) and "`.`" (current directory).

To change the current directory, see section "4.6 Change Directory" on page 15.

4.3 Read

There are two types of optical disc read APIs: synchronous, meaning the function does not return until the read finishes; and asynchronous, meaning the function returns immediately while the read occurs in the background.

4.3.1 Synchronous Read

Code 4–3 DVDRead()

```
#include<revolution/dvd.h>

s32  DVDRead( DVDFileInfo* fileInfo, void* addr,u32 length, u32 offset);
```

`DVDRead()` reads a portion of the file and stores it into the buffer specified by *addr*. *Length* and *offset* specify which part of the file to read, where *offset 0* means the top of the file.

`DVDRead()` returns the number of bytes transferred if the read succeeded; it returns –1 if the read did not.

The programmer must prepare enough memory space for the buffer. To get the size of the file, use the `DVDGetLength()` macro (see "4.5 Get Size" on page 15).

Because of drive hardware restrictions, *addr* should be 32-byte aligned, *length* should be a multiple of 32, and *offset* should be a multiple of 4.

As noted above, `DVDRead()` does not return until the read finishes, hence the term “synchronous.” Therefore, if a user opens the cover, `DVDRead()` will keep waiting in the function for the user to close the cover. In the meantime, the CPU executes other runnable threads, if any. For more information on threads, refer to the *Operating System*.

Invalidating the data cache is unnecessary because `DVDRead()` does this by default. To change this behavior, call `DVDSetAutoInvalidation()` (described in the online *Revolution Function Reference Manual*). If the read data contains a program, however, you must invalidate the instruction cache regardless of whether you have changed this default behavior.

4.3.2 Asynchronous Read

Code 4–4 DVDReadAsync()

```
#include<revolution/dvd.h>

typedef void (*DVDCallback)(s32 result, DVDFileInfo* fileInfo);

BOOL DVDReadAsync(DVDFileInfo* fileInfo, void* addr, u32 length,
       u32 offset, DVDCallback callback );
```

`DVDReadAsync()` issues a command to the drive to read the file into the buffer specified by *addr* and then returns immediately. *Length* and *offset* specify which part of the file to read, where *offset 0* means the top of the file. The function specified by *callback* is invoked when the read finishes; however, you can set *callback* to `NULL` if you do not want any callback to be invoked.

As with `DVDRead()`, you must prepare enough memory space for the buffer. To get the size of the file, use the `DVDGetLength()` macro (see "4.5 Get Size" on page 15).

And again, *addr* should be 32-byte aligned, *length* should be a multiple of 32, and *offset* should be a multiple of 4.

`DVDReadAsync()` returns `TRUE` if it issued the command successfully, and `FALSE` if it did not.

The argument *result* shows whether the read succeeded or not. If the read was successful, the number of bytes transferred is set; the function returns –1 if the read was not successful.

“Asynchronous” means that the read goes on in the background. Therefore, you should call `DVDGetFileInfoStatus()` to display the appropriate message to the user while the transfer executes in the background. (See "5 Error Handling" on page 16 for more details about error handling.)

`DVDReadAsync()` invalidates the data cache by default. To change this behavior, call `DVDSetAutoInvalidation()` (described in the online *Revolution Function Reference Manual*). If the read data contains a program, however, you must invalidate the instruction cache regardless of whether you have changed this default behavior.

You can call as many read commands as you want before any previously issued read finishes. In such cases, the argument *fileInfo* of *callback* shows which file transfer has just finished.

The following code sample reads a file using `DVDReadAsync()`:

Code 4–5 Asynchronous File Read

```
1:      // This variable should be declared "volatile" because this is shared
2:      // by two contexts.
3:      volatile s32     readDone = 0;
4:
5:      // The name of the file we are going to read.
6:      char*   fileName = "testfile1.txt";
7:
8:
9:      static void callback(s32 result, DVDFileInfo* fileInfo)
10:     {
11:         if (result == -1)
12:         {
13:             readDone = -1;
14:         }
15:         else
16:         {
17:             readDone = 1;
18:         }
19:         return;
20:     }
21:
22:     void main(void)
23:     {
24:         DVDFileInfo fileInfo;
25:         u32         fileSize;
26:         u8*         buffer;         // pointer to the buffer
27:
28:         MyOSInit();
29:
30:         DVDInit();
31:
32:         // We MUST open the file before accessing the file
33:         if (FALSE == DVDOpen(fileName, &fileInfo))
34:         {
35:             OSHalt("Cannot open file");
36:         }
37:
38:         // Get the size of the file
39:         fileSize = DVDGetLength(&fileInfo);
40:
41:         // Allocate a buffer to read the file.
42:         // NOTE: Pointers returned by OSAlloc are always 32byte aligned.
43:         buffer = (u8*)OSAlloc(OSRoundUp32B(fileSize));
44:
45:         // This function only start reading the file.
46:         // The read keeps going in the background after the function returns
47:         if (FALSE == DVDReadAsync(&fileInfo, (void*)buffer,
48:                                   OSRoundUp32B(fileSize), 0, callback))
49:         {
50:             OSHalt("Error occurred when issuing read");
51:         }
52:
53:         while (1)
54:         {
55:             switch (readDone)
56:             {
57:               case 1:       // the read succeeded
58:                 goto exit;
59:                 // NOT REACHED HERE
60:               case -1:      // the read failed
```

```
61:                 OSHalt("Error occurred when reading file");
62:                 // NOT REACHED HERE
63:               case 0:       // processing...
64:                 // do something
65:                 break;
66:             }
67:         }
68:
69:       exit:
70:         // From here, we can use the file
71:         OSReport("read end\n");
72:
73:         // Close the file
74:         DVDClose(&fileInfo);
75:
76:         :
77:
78:         OSFree(buffer);
79:
80:         OSHalt("End of demo");
81:
82:         // NOT REACHED HERE
83:     }
```

You can do anything (for example, show graphics, play audio, and so forth) while reading the file in the background. If the read finishes, *callback* is invoked. You can write whatever you want in the callback function. In this example, we wrote it to set the flag *readDone* so that the main loop (lines 53-67) can tell whether the read finished.

Note: *Volatile* is necessary for the variable *readDone* because *readDone* is accessed by two contexts, `main` and *callback*. Without *volatile*, compilers might assign a register for the variable, which will cause inconsistency between the two contexts.

The above code is stored as "`dvddemo/src/dvddemo2.c`." Try it to see how it works.

Although we didn't document the code here, `dvddemo3` shows you how the system works when you issue a `DVDReadAsync()` without waiting for the previously issued read command to finish.

Note: You *cannot* issue `DVDReadAsync()` with a *fileInfo* that has already been used in a previously issued and/or unfinished `DVDReadAsync()` argument. For example, you might want to write something like the following code segment to read each half of the file separately, but this is illegal.

Code 4–6 Example of Illegal Code

```
DVDReadAsync(..., fileInfo, ...);
// Illegal!!
DVDReadAsync(..., fileInfo, ...);
```

In this next example, the second `DVDReadAsync()` fails and returns `FALSE`. To solve this problem, you have two options. Either you wait for the first read to finish, like this:

Code 4–7 Legal Code, Option 1

```
DVDReadAsync(..., fileInfo, ...);
:
// Wait to finish the first read
:
DVDReadAsync(..., fileInfo, ...);
```

Or you use a different *fileInfo* for each part, like this:

Code 4–8 Legal Code, Option 2

```
DVDOpen(..., fileInfo1);
DVDOpen(..., fileInfo2);

DVDReadAsync(..., fileInfo1, ...);
DVDReadAsync(..., fileInfo2, ...);
```

It is legal to open one file with multiple *fileInfo*s.

4.4 Close

Code 4–9 DVDClose()

```
#include<revolution/dvd.h>

BOOL DVDClose(DVDFileInfo* fileInfo);
```

If you no longer need to access the file, close the file with `DVDClose()`. If *fileInfo* is in use (i.e., the `DVDReadAsync()` function using *fileInfo* hasn't finished yet), `DVDClose()` returns `FALSE`.

4.5 Get Size

`DVDGetLength()` gets the size of the file specified by *fileInfo*. This is useful when you want to allocate memory to read the data dynamically.

Code 4–10 DVDGetLength()

```
#include<revolution/dvd.h>

u32 DVDGetLength(DVDFileInfo* fileInfo);
```

4.6 Change Directory

Code 4–11 DVDChangeDir()

```
#include<revolution/dvd.h>

BOOL DVDChangeDir(char* dirName);
```

This function changes the current directory. As you can imagine, if you specify the directory beginning with a "`/`" (as in "`/texture`"), the function will try to change the current directory to"`/texture`" as an absolute path name. If you specify the directory beginning without a "`/`" (as in "`texture`"), the function treats the directory as relative to the current directory.

If the function succeeded in changing directory, it returns `TRUE`; otherwise, it returns `FALSE`.

5 Error Handling

The Revolution optical disc API provides a very simple way to handle errors.

Code 5–1 DVDGetFileInfoStatus()

```
#include<revolution/dvd.h>

u32 DVDGetFileInfoStatus(DVDFileInfo* fileInfo);
```

This function checks the transfer status for *fileInfo* and returns one of the following values:

Table 5–1 Error Messages Returned by DVDGetFileInfoStatus

Definition	Value	Meaning
DVD_STATE_FATAL_ERROR	–1	A fatal error occurred.
DVD_STATE_END	0	Transfer finished/no transfer requested.
DVD_STATE_BUSY	1	Transfer is being processed.
DVD_STATE_WAITING	2	Transfer is queued and waiting to be processed.
DVD_STATE_NO_DISK	4	No optical disc in the drive.
DVD_STATE_WRONG_DISK	6	Wrong optical disc in the drive.
DVD_STATE_MOTOR_STOPPED	7	Drive's motor stopped.
DVD_STATE_CANCELED	10	Transfer was cancelled.
DVD_STATE_RETRY	11	A retry error was generated.

Note: The DVD_STATE_COVER_OPEN, DVD_STATE_COVER_CLOSED, and DVD_STATE_IGNORED errors messages that existed in Nintendo GameCube were deleted.

Here is some sample pseudo-code for how to use this function:

Code 5–2 Error Handling

```
:
DVDReadAsync(fileInfo);
:
while(1)
{
    :
    // Foreground process
    :
    switch (DVDGetFileInfoStatus(fileInfo))
    {
      case DVD_STATE_FATAL_ERROR:
        // Display “Fatal error occurred”
        // Stop the game
        break;

      case DVD_STATE_END:
        // File is read successfully
        break;

      case DVD_STATE_BUSY:
        // Display "Now loading..."
        break;

      case DVD_STATE_NO_DISK:
        // Display “Open the cover and put in the right disc”
        break;

      case DVD_STATE_WRONG_DISK:
        // Display “This is not a disc of game XXX. Replace it to the right disc”
        break;
      :
      :
    }
}
```

Note: The `DVDRead()` and `DVDReadAsync()` functions process the following tasks internally:

- Wait for the user to open the cover and then close it (when no optical disc or wrong disc detected)
- Read a certain area, which is called the optical disc ID, to verify if the correct disc is in the drive
- Issue the read function again if the correct optical disc is verified to be in the drive

You need only to check the return value of `DVDGetFileInfoStatus()` and display the appropriate message to the user. No complicated error handling programming is needed.

6 File System

This chapter briefly discusses how the Nintendo GameCube file system works and discusses some tips for developers.

The Nintendo GameCube file system uses a conversion table called FST (for File Symbol Table). Every time `DVDOpen` is called, Nintendo GameCube searches for the filename in the FST and retrieves the file information from it. In this chapter, we will mainly discuss the following:

- Size of the FST
- Location of the FST
- Speed of conversion
- Character set usable for file/directory names

6.1 FST Size

FST size can be calculated easily from the following equation:

Equation 6-1 Calculating FST File Size

$$12 + (\textit{length of filename} + 1)\ (\textit{bytes} / \textit{entry})$$

So, if we have a file named "`foo`," it consumes 12 + 3 + 1 = 16 bytes in the FST (the last +1 is for the termination `NULL`).

The same logic can be applied to directories. If we have a file named "`foo`" under "`/baa1/baa2/`", and there are no other files or directories on the optical disc, the size of the FST is going to be:

Equation 6-2 Calculating FST Directory Size

$$12[\textit{root directory}] + (12 + 4 + 1)\ [\textit{baa}1] + (12 + 4 + 1)\ [\textit{baa}2]\ (12 + 3 + 1)\ [\textit{foo}] = 62[\textit{bytes}]$$

6.2 FST Location

The boot program loads the FST at the highest memory point, higher than *arenahi*. Therefore, using Equasion 6-2, which uses 62 bytes for FST, the FST is loaded to 0x817f_ffc0 (i.e., 24MB – 64 bytes; note that the 32-byte alignment) and *arenahi* is set to the same location.

6.3 TIPS to Reduce FST Size and Speed

In general, having fewer files reduces the conversion speed. Changing the directory is also a good idea. Since `DVDOpen` starts searching for the file in the current directory, changing the directory to that of the target file before you call `DVDOpen` reduces the conversion speed if you need to open more than one file.

For example, say you need to open two files, "`/foo/baa/dummy1`" and "`/foo/baa/dummy2`". If you simply call `DVDOpen` twice, `DVDOpen` will need to search two separate times from the beginning of the optical disc. However, if you call `DVDChangeDir` to change the current directory to "`/foo/baa`", the two subsequent `DVDOpen` calls using relative path names will be faster, even allowing for the time it takes `DVDChangeDir` to search the directory (it uses the same algorithm as `DVDOpen`).

6.4 Character Set Usable for File/Directory Names

You can use 7-bit ASCII characters for file/directory names, except the following:

- " * , / : ; < > ? \ |
- Control codes

In other words, you can use the following characters:

- Numerals
- Alphabets
- Spaces
- ! # $ % & ' ( ) + - . = [ ] ^ _ @ ` { } ~

Examples of *unusable* characters include Kanji, “Hankaku kana,” Japanese letters, and accents (such as the “é” in “Pokémon”).

Note: File and directory names are case-insensitive. For example, `foo.txt` and `FOO.TXT` and `FoO.tXt` would all be considered the same file. You can use *any* combination of lowercase and capital letters to open files and directories.

There is no restriction on the length of file and directory names. As far as the tool (or to be more precise, Windows) allows, you can use as long name as you want.

7 Directory Access Support Functions

Sometimes you may want to read all of the files in a certain directory without knowing the name of each file. Or you may want to make common functions which change their behavior according to the files under a directory. To facilitate directory access, we've provided BSD UNIX-like directory access functions.

In most cases, you can write code using three functions, open, read, and close, to access directories. So first we'll discuss how to use the three functions, and then introduce a simple code sample that uses these three functions. After that, we'll look at other useful functions.

7.1 Open Directory

Code 7–1 DVDOpenDir()

```
#include<revolution/dvd.h>

typedef struct
{
    u32         entryNum;
    u32         location;
    u32         next;
} DVDDir;

BOOL DVDOpenDir(char* dirName, DVDDir* dir);
```

`DVDOpenDir()` opens a directory specified by *dirName* and associates *dir* with that directory. It returns `TRUE` if it found the directory; otherwise, it returns `FALSE`.

You can use both relative and absolute path names for *dirName*. If you use a relative path name (i.e., *dirName* starts without a "/"), it will mean relative to the current directory.

7.2 Read Directory

Code 7–2 DVDReadDir()

```
#include<revolution/dvd.h>

typedef struct
{
    u32         entryNum;
    BOOL        isDir;
    char*       name;
} DVDDirEntry;

BOOL DVDReadDir(DVDDir* dir, DVDDirEntry* dirent);
```

`DVDReadDir()` gets information on the next directory "entry." The directory entry is either a directory or a file. If it's a directory, `dirent->isDir` is `TRUE`. You can call `DVDOpenDir()` to open the subdirectory using dirent->name. If it's a file, `dirent->isDir` is `FALSE`. You can call `DVDOpen()` to open the file using `dirent->name` as well.

The function returns `FALSE` upon reaching the end of the directory or detecting an invalid location was set by `DVDSeekDir()`.

7.3 Close Directory

Code 7–3 DVDCloseDir()

```
#include<revolution/dvd.h>

BOOL DVDCloseDir(DVDDir* dir);
```

`DVDCloseDir()` closes the specified directory structure. It returns `TRUE` on success; otherwise, it returns `FALSE`.

7.4 Sample Code for Accessing a Directory

This is a very simple sample code to show how to access a directory. This code opens a directory "`foo/baa`" (relative to the current directory) and show all the files and directories under there.

Code 7–4 Sample Code for Accessing a Directory

```
    DVDDir          dirFooBaa;
    DVDDirEntry     dirent;

    if (FALSE == DVDOpenDir("foo/baa", &dirFooBaa))
    {
        // print "can't open the directory" and exit
    }

    while(TRUE == DVDReadDir(&dirFooBaa, &dirent))
    {
        OSReport("name: %s, type: %s\n", dirent.name,
                 (dirent.isDir)? dir : file);
    }

    if (FALSE == DVDCloseDir(&dirFooBaa))
    {
        // print "can't close the directory" and exit
    }
```

7.5 Tell Directory

Code 7–5 DVDTellDir()

```
#include<revolution/dvd.h>

u32 DVDTellDir(DVDDir* dir);
```

`DVDTellDir()` is a macro that returns the current location of the specified directory structure.

This function should be used with `DVDSeekDir()` to provide "save" and "restore" features for directory access. You can use `DVDTellDir()` to get the current location of the directory structure, then restore it using `DVDSeekDir()` to read the directory from the previous location again.

7.6 Seek Directory

Code 7–6 DVDSeekDir()

```
#include<revolution/dvd.h>

void DVDSeekDir(DVDDir* dir, u32 loc);
```

`DVDSeekDir()` is a macro that sets the position of the next `DVDReadDir()` on the directory structure.

This function should be used with `DVDTellDir()` to provide “save” and “restore” features for directory access. It’s dangerous to use a number for *loc* that is not returned by `DVDTellDir()`. It’s also dangerous to use a number for *loc* that is returned by `DVDTellDir()` but for the other directory structure.

Note: This function is a macro and performs no argument checking, but if you happened to specify an invalid value for *loc*, `DVDReadDir()` can detect it and return `FALSE`.

7.7 Rewind Directory

Code 7–7 DVDRewindDir()

```
#include<revolution/dvd.h>

void DVDRewindDir(DVDDir* dir);
```

`DVDRewindDir()` resets the position of the specified directory structure to the beginning of the directory. A call to `DVDReadDir()` after this function behaves as if the directory had just been opened.

8 Optical Disc Drive FAQs

1. Is the data recorded in one spiral, similar to CDs?

Yes.

2. How fast are the optical disc drive's random-access capabilities?

Compared to ROM cartridges, which have very fast seek times, we must accept that the optical disc drive will have a poorer random-access capability. To minimize any negative impact, you should place data as often as possible in the order that the game will need it. You can also solve this problem by duplicating data many times on the optical disc, but of course this trades data capacity for speed.

3. How many files can I open at one time?

There is no limit; you can open as many files as you want.

4. Can I call DVD*() from within a callback?

Yes. For example, you can call `DVDClose()` when the read of the file finishes, like this:

Code 8–1 FAQ Code Sample

```
void callback(s32 result, DVDFileInfo* fileInfo)
{
    if (result == DVDGetLength(fileInfo))
    {
        DVDClose(fileInfo);
    }
    ...
}
```

As another example, you can call `DVDReadAsync()` from within an audio callback routine while accessing audio data.