Nintendo GameCube™ Memory Card Guidelines

Version 2.1

Table of Contents

DOL-06-0025-001-A5
Released: 1/17/02

Table of Contents (Continued)

Revision History

Version	Revision Date	Description of Revisions
2.1	12/21/01	- Added new paragraph at end of 3.3
2.0	11/30/01	- Revised paragraphs 3.5, 4.3, 6.4, Figure 1, and Figure 2 - Added paragraph 3.6
1.8a	10/16/01	- Paragraph 3.8, Added to descriptions and added sample messages for clarification - Paragraphs 6.4 and 6.5, Revised messages
1.8	9/12/01	- Revised paragraphs 3.2, 3.4, 3.5, 3.6, 3.7, 3.8, 6.3, 6.4, and 6.5
1.3	8/3/01	- Revised paragraphs 2.2, 2.3, 3.3, 3.4, 3.8, 3.9, 3.10, 3.11, 6.2, and 6.16 - Added paragraphs 6.3, 6.17, and 6.18
1.0	7/13/01	- Released by NOA

1. Introduction

This document explains the Nintendo GameCube™ Memory Card.

The Memory Card is a backup medium for Nintendo GameCube™ applications. It is used to store saved game data, high scores and other information for the game software. It can be attached and detached from the Nintendo GameCube™ Memory Card Slot (Slot A and Slot B) on the front of Nintendo GameCube™.

This document provides a summary of the Memory Card as well as some warnings concerning the creation of programs. The "Game Cube Function Reference Manual" contains details about the various APIs, so please look at the reference manual and this document together.

2. The Memory Card

2.1 Warnings about flash memory

The Memory Card is a backup medium for the Nintendo GameCube™. A flash memory device is mounted in the card as the data storage IC. Flash memory has the following characteristics. Please take note of these features when creating your programs.

- Reading is fast, but writing is slow.
- Card performance deteriorates over repeated write cycles and eventually writing will become impossible.
- Reading/writing cannot be performed in 1byte units, and only in certain specific units.

2.2 The file system

The Memory Card has a simple file system, and applications must use the library to perform reads/writes to the flash memory inside the card. For example, to store the saved game data, the application follows a procedure in which it first creates a file inside the flash memory and then writes the data to that file. The application cannot ignore the file system and freely read/write to the flash memory inside the card.

The Memory Card capacity is 512Kbyte (4Mbit). The file system manages this capacity by separating it into units with 8Kbyte boundaries. This 8Kbyte smallest unit of memory is called a sector. In other words, files need to be created in units of sectors.

Since the Memory Card capacity is 512Kbyte, there are 64 sectors. However, the file system uses five of these sectors, so the application can make use of a maximum of 59 sectors.

For users, the term "block" is used when referring to sectors. Thus, a 4Mbit Memory Card has 59 usable blocks of memory.

The values used in the above paragraphs are for Memory Cards made to the current set of specifications. Memory Cards of larger capacity may be sold in the future, in which case the capacity and block size, etc. may change. For details, please see "4. Large-capacity Memory Cards."

Memory Cards shipped from the factory are already formatted.

2.3 Memory Card Screen

The console Initial Program Loader (IPL) Memory Card screen is a menu program for the operation of Memory Card files. It is included in the Nintendo GameCube™ console's built-in IPL. This menu program has the following functions:

- Display a list of the files in the Memory Card
- Delete files
- Move files
- Copy files
- Initialize Memory Card (only a Memory Card with destroyed contents can be formatted.)

It also has specifications to display the following information so users can identify individual files:

- Comments (1st line: Game name, 2nd line: detailed explanation of saved data)
- Icons (32x32, CI8 or RGB5A3; can be animated)
- Banners (96x32, CI8 or RGB5A3)

Comments and Icons must be prepared for an application that uses the Memory Card. Banners are not required and can be ignored, but we recommend using them. When they are ignored, Icons will be displayed in the banner display column.

Creation of a Memory Card menu to serve as the application's personal Memory Card Screen is prohibited. Any Memory Card memory menu you create inside the application should only be designed to handle files that are used by that application.

2.4 Memory Card of overseas specifications

There are 2 types of Memory Cards, Kanji type (for Japan) and Alphabet type (for countries other than Japan). The only difference between the two is their software format. There is no hardware difference.

Memory Cards of Kanji type are formatted so that character codes of Japanese shift JIS (first level) are used. On the other hand, Memory Cards of Alphabet type are formatted so that ANSI8 bit (WinLatin1) are used.

Memory Cards are formatted for a particular market and cannot be read when inserted into a Nintendo GameCube™ system from another market.

2.5 Flash memory hardware specifications

As a reference source, this section explains the hardware specifications for the flash memory used in the Memory Card.

Capacity: 4Mbit (512Kbyte)

Sector size: 8192 bytes

Page size: 128 bytes

Speed: Refer to the table below

Parameter	Typical Value	Maximum Value
Sector Erase Time	200ms	1600ms
Page Write Time	2ms	60ms

These are catalog spec values. In reality, it is considered faster than the typical value noted above. Please note that there are individual differences.

3. Using the Memory Card

3.1 API

A CARD API has been prepared as an API for the Memory Card. For details about this CARD API, read the introduction to the Memory Card section in the "Nintendo GameCube Function Reference Manual", as well as the explanation for individual APIs.

3.2 Recognizing Memory Cards

Due to the chattering operation, the `CARDProbe` and `CARDProbeEX` functions may not return TRUE immediately, even if a Memory Card is inserted. In this context, chattering refers to the detection of multiple input signals when the Memory Card is inserted. Include a process on the save screen that waits for `CARDProbe` and `CARDProbeEX` to return TRUE. For example, prepare a screen that indicates the status of Slot A and Slot B, and if `CARDProbe` and `CARDProbeEX` return TRUE, highlight the slot. If FALSE is returned, gray out the slot and make grayed out slots impossible to select for saving data.

Once the CARDProbe and CARDProbeEX functions return FALSE, do not immediately display an error message. The CARDProbe and CARDProbeEX functions require very little processing time, so there is no overhead, even if the application calls the function every frame.

3.3 Working with Slots

There are two slots on the Nintendo GameCube™ into which Memory Cards can be inserted -- Slot A and Slot B. Memory Card-ready applications should work no matter which slot users choose for inserting their Memory Card. This is for the convenience of users, since one of the slots may be occupied by a device other than a Memory Card.

We recommend that the application support both slots, but if the application is only going to support one slot, you must display a message to the user so they know which slot to use for the Memory Card.

If the application supports both slots, display a message to the user so they know which slot will be accessed. For example, when saving display a message similar to the following, “Data will be saved to the Memory Card in Slot A.” If the slot being accessed is not specified, for example, “Data will be saved to the Memory Card.”, the user may not be able to determine which Memory Card was used to save the data.

3.4 Creating files

The following items must be specified, when creating a file for the Memory Card.

Filename The filename is the name used by the API to specify the file. The maximum byte size is given by `CARD_FILENAME_MAX` (32bytes). Even if two files have the same filename, they can be distinguished if the application's company code, game code or game version differ. For this reason, applications can freely name files.

File size The size of the file must be specified as an integer multiple of the block size (8Kbytes).

Icon The icon is the image used on the Nintendo GameCube™ Memory Card Screen. It can be up to 8 frames of animation. The size is 32x32. Select from two formats for the icon: CI8 format or RGB5A3 format. When you use CI8 format, the color index is shared. The icon animation speed can be set to fast, medium or slow. Two types of icon animation patterns can be set. For example, if the icon is animated with three frames of animation (1,2,3) the pattern can be set to repeat from the beginning (123123123...) or to loop back and forth (123212321…). You can also change the icon based on the game progression.

Banner The banner is the text string used to display comments in the Nintendo GameCube™ Memory Card Screen. The size is 96x32. Select either CI8 format or RGB5A3 format. It is possible to omit the banner if there is a reason, such as to economize capacity. In such cases, the icon will appear in the frame where the banner normally displays in the Memory Card menu. You can also change the banner based on the game's progression.

Comments Comments is the text string used to display comments in the Nintendo GameCube Memory Card Screen. The characters that can be used are the same as the fonts that can be used for FONT API. For the Kanji-spec (Japanese) Memory Card, the text can be ASCII text, single-byte katakana, and Shift JIS Level One. For the Alphabet-spec (non-Japanese) Memory Card, the characters can be ANSI 8-bit (WinLatin1). The Comments can be a 64-byte character string, but this is displayed in the file menu, divided into two lines with 32 bytes on each line. Enter the game name on the first line, and use the second line to give an explanation of the file. For a game that will have many files, please enter the same character string for the game name on the first line each time. Also, please note that if you use a great deal of characters with a wide character width, the text may not display completely in the Memory Card Screen's comments area. File names and descriptions must fit completely in the Memory Card Screen.

The icon, banner and comments are recorded within the file, and not in the system block. The icon data should be placed below file offset 512 bytes.

Capacity for each is shown in the table below.

	Required capacity (for single icon)		Required ?
Icon	RGB5A3 2Kbytes	CI8 1.5Kbytes	Required
Banner	RGB5A3 6Kbytes	CI8 3.5Kbytes	Recommended
Comment	64bytes		Required

Since data such as icons is recorded within the file, a file size becomes larger if icons with more data are recorded. For this reason, try to avoid situations where the icon data alone makes the file extremely large even when there is only a small amount of actual data stored in a file.

For icons prepared in RGB5A3 format, the size of each frame is 2Kbyte, so an 8-frame animation alone takes up 16Kbytes. If the banner data is also stored in RGB5A3 format, then the icon and banner together will fill 22Kbytes. In this case, the file size will be 3 blocks even if it contains only several dozen bytes of actual game data. As a result, the Memory Card will not be able to store very much actual data and the user will feel cheated.

The way to economize the most on icon and banner size is to use a single icon CI8-format and also use the CI8 format for the banner. But even this takes up around 5Kbyte including the Comments area. In other words, a 1-block file has less than 3Kbytes of space for the actual recording of data. Of course, the banner can be omitted, in which case a 1-block file has room for around 6.5Kbytes of data.

Also, when displaying a file's comments during game-play (such as in a Save menu), use the font library. You can use your own fonts, instead of the font library. However, if you do use your own fonts, take special care to ensure that unanticipated problems do not occur, such as characters that are missing from your fonts and therefore do not display.

3.5 Checking the amount of free space

Timing is critical when checking for the amount of free space. If you give no special consideration to checking for free space and only check when it is time to save data, the following problem can arise.

> The user starts the game from the beginning. After playing the game for an extended period the user elects to save the game, figuring it is around time to take a break and assuming he or she will be able to continue on from that point tomorrow. But a message says that there is not enough space and the data cannot be saved. The user has no choice but to turn off the power and lose their progress.

In this scenario, the application checks for free space when it was actually time to save the game, but by then it is too late. If the application had checked for free space before the user even started the game, then this problem could have been avoided. An appropriate message should be displayed prior to game play, when there is not enough space to save the game.

Specifically, a message saying, "There is not enough free space on the Memory Card. In order to save this game, YY free files and XX free blocks are required" should be displayed when there are not enough free blocks. A message saying "Since the number of files will exceed the limit, no additional files can be created. To save it requires YY free files and XX free blocks." should be displayed when there are not enough free file entries.

You should first check the free blocks, then check the free file entries. This considers the case when there are no free file entries or free blocks. If you check them in this order, a message saying "XX free blocks are required." is displayed when there are no free file entries or free blocks. If the order of checking is reversed, the necessary number of blocks cannot be displayed. This would be undesirable for users, since they could not determine what size file to delete.

Alternatively, you could use the same message both when there are not enough free blocks and when there are not enough free files. If you choose to handle the error message like this, display a message such as, "There is not enough free space on the Memory Card in Slot A, or the number of files will exceed the limit. To save this game, you need at least 1 free file and XX blocks."

Also, the `OSResetSystem` function is available for use in supporting users who want to perform a reset on their Nintendo GameCube™ to call up the Memory Card Screen, in order to organize the contents of their Memory Card. You should provide access to the Memory Card Screen when there is not enough space available on the Memory Card. If you do not include this functionality in your game, display a message saying, "To manage the contents of your Memory Card, use the Memory Card Screen."

The process flow is illustrated in Figure 1. This flowchart is only for reference, and there is no need to follow the same process flow or use the same messages.

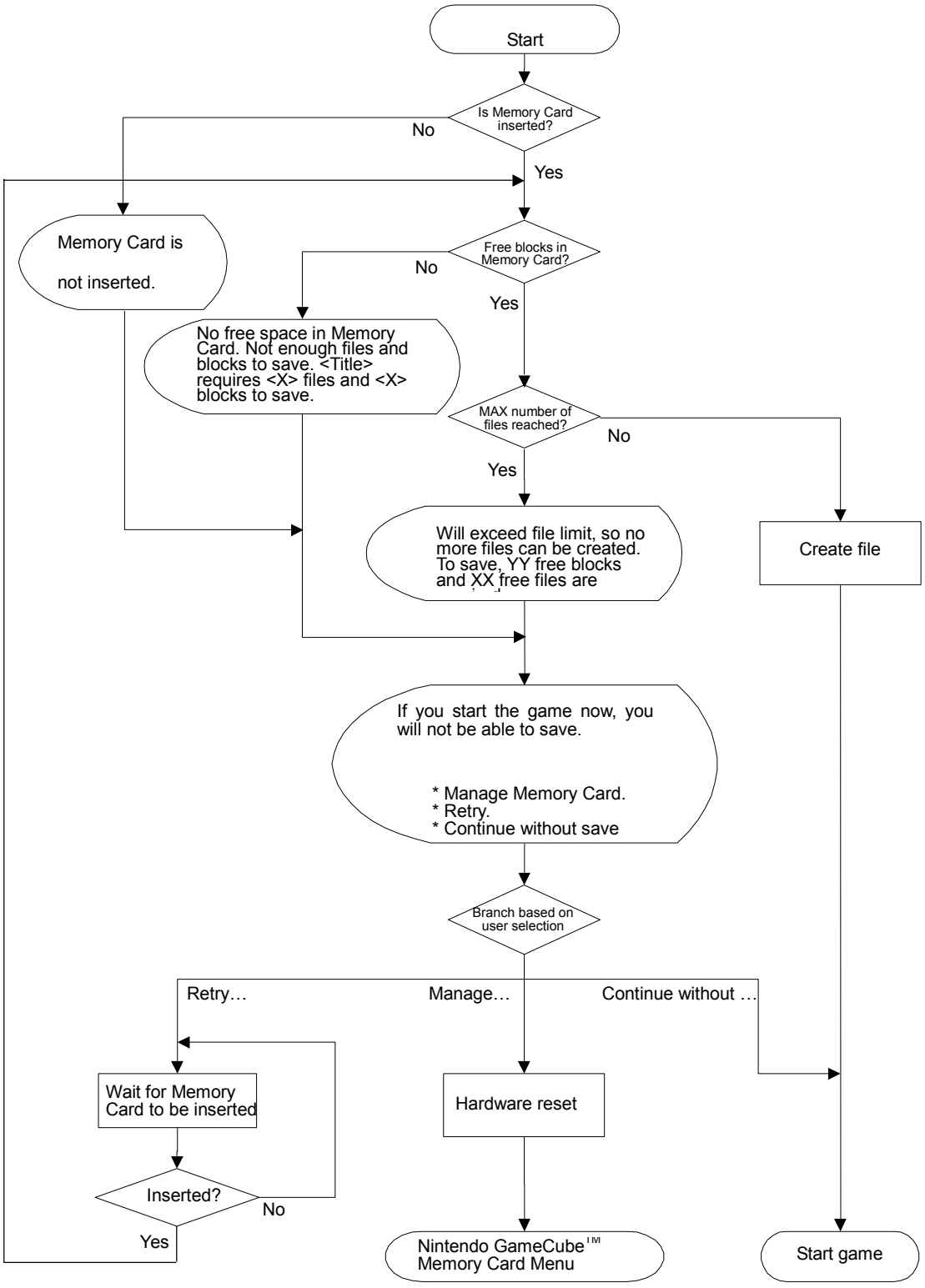

Figure 1 Check free space when starting game

3.6 Inserting devices other than Memory Cards

In the future, there is the possibility that the Memory Card slots will be used for devices other than Memory Cards. Therefore, it is possible that another device remains inserted when you try to use a Memory Card. In this case, it would be helpful to users if some kind of caution is displayed. If an error, `CARD_RESULT_WRONGDEVICE` is returned when the function `CARDMount` or `CARDMountAsync` is called, a message saying “Another device in Slot <A or B>. Please insert Memory Card.” should be displayed on the screen.

3.7 Memory Card formatting issues

When a Memory Card that is formatted for another market is inserted into the Nintendo GameCube™, formatting is performed after the game player is contacted.

Specifically, in case an error `CARD_RESULT_ENCODING` is returned when functions `CARDMount` or `CARDMountAsync` are called, a message such as "The Memory Card in Slot <A or B> contains corrupted data and needs to be formatted. Do you want to format?" should be displayed on the screen. It should perform formatting only when a user replies "Yes".

3.8 Repairing the Memory Card

The Memory Card hardware and the library both support the insertion and removal of Memory Cards during game play. However, there is the possibility of Memory Card data being destroyed if the Memory Card is removed or the power is removed while writing data. Problems can arise if either happens while reading data as well. To avoid these problems, messages should be displayed on the screen warning the user not to turn off the power or remove the Memory Card.

For example, before starting access to the Memory Card, you should display a message saying something like "Ready to access. Do not touch the Memory Card or the POWER Button" and then wait for button input from the user. During access display a similar message like "Accessing. Do not touch the Memory Card or the POWER Button."

If by chance the contents of a file on the Memory Card are destroyed, the system itself will not be able to detect that damage. So please design the application to check the consistency of the data.

The chances of this are slim, but in the worse case scenario the system region of the Memory Card could itself be destroyed. It is thus also necessary for the application to be able to deal with the insertion of a damaged Memory Card. As the basic process, call the function `CARDCheck` or function `CARDCheckAsync` when `CARD_RESULT_READY` and `CARD_RESULT_BROKEN` are returned after the Memory Card has been mounted. These check the consistency of the file system, and if it is damaged, try to repair it. If calling these functions cannot repair the Memory Card, then take the following steps.

First, ask the user if he/she desires to format the Memory Card with a message like the following: "The Memory Card in Slot <A or B> contains corrupted data and needs to be formatted. Do you want to format?". Format the Memory Card only if the user agrees to do so. If this does not correct the problem (format was unsuccessful), display a message like the following, "The Memory Card in Slot <A or B> is damaged and cannot be used."

To create a broken Memory Card, as of the date of this release, you need to download memcard_erase.zip from the Nintendo of America Software Development Support Group website www.noa-engineering.com . In the future, this tool will be included in the Nintendo GameCube™ Software Development Kit (SDK).

Figure 2, on the following page, illustrates the process flow when a Memory Card is mounted, including efforts to repair damage.

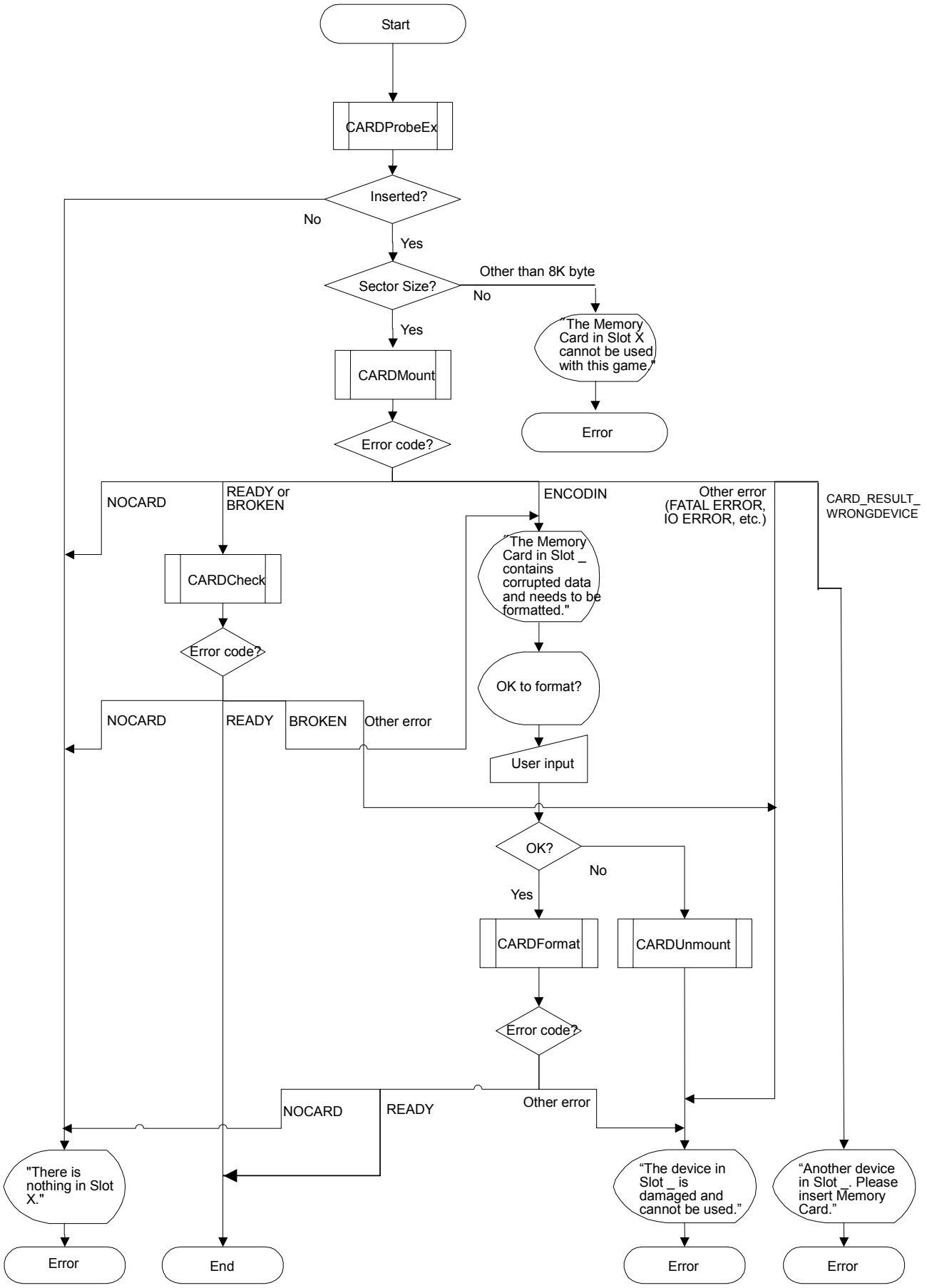

Figure 2. Flow chart when mounting, including repairing Memory Card

3.9 Cautions when writing data

As mentioned above, if the Memory Card is removed while writing data the integrity of previously saved data cannot be guaranteed. For this reason, please adopt a method in the writing process that can verify that the data will be normal when read. Then, at reading time, perform the verification process and, if the data is corrupted, display a message saying something like "Save data has been corrupted." No measures are taken, so be careful that the application does not run with the corrupted data and produce unexpected results.

The easiest and most effective way to check that files are normal is to use a checksum or CRC. Furthermore, it is effective to check to see if each data is a normal value that can be used in the application, using a program.

There may be some situations in which you cannot ensure the integrity of files with only a checksum. For example, problems could occur when the power is turned off after a file has been deleted and a new file created, but before the initialization data has been written to the Memory Card. In such situations, the old data may exist in the file's data area and, when the file is read, the old data that was supposed to have been erased could be loaded instead of the new data. In order to prevent this situation from occurring, we recommend the following procedure:

1. Use `CARDCreate` to create the file. At this point, the file does not contain an icon or comments.
2. Use `CARDWrite` to write the icon, comments, and initialization data.
3. Use `CARDSetStatus` to specify the icon and comment's offset value.

When opening files created using the procedure described above, always use `CARDGetStatus` to check that the icon and comment's offset value is correct. The offset value for the icon and comment are saved in the file's directory area, and is set to `-1` when a file is created. After confirming that the icon and comment's offset value is correct, load the data and verify that the data area is normal using a function such as checksum.

For essential data, you can use any of the three following processes to help prevent the destruction of data to some extent, even when the Memory Card is removed during writing.

1. In the first method, the data is duplicated. The important parts of the file are written in two places, making a backup. When adopting the method, be careful not to duplicate the data inside the same block, as that would be meaningless. The "blocks" in the Memory Card file system are sectors in the flash memory. When data is written to one block, the contents of that block are first completely erased, and then the data is written. So if the dual data is held in the same block, both copies will be erased at the time of writing. Thus, if you are going to duplicate data, be sure to place the backup in a separate block. The drawback to this method is that it consumes excessive capacity.

   When using this method, if the number of free files is <X and/or the number of free blocks is <Y a message similar to the following must appear:

   For example:

   "There is not enough space on the Memory Card in Slot <A or B>. <Title> requires <X> files and <Y> blocks to save. [Retry] [Continue without save] [Memory Card Screen]"

2. In the second method, data is not written directly to the original file, but instead to separate file using the following procedure: First, the Memory Card is checked to verify that there is enough space for another file. If so, a different file is created and the newest data is written there. Once that write operation is completed, the original file is deleted. By following this procedure, the original file will not be damaged even if the Memory Card is removed during writing. If there is not enough free space on the Memory Card, then the data is written directly to the original file. The thing to be careful about here is the removal of the Memory Card before the original file can be deleted, resulting in two files. If you are going to use this method, be sure to design the program so it can operate normally even if two files exist. The drawback to this method is that it offers no added safety if there is not enough free space on the Memory Card. Its primary merit is efficient use of blocks.

   Using this method, we duplicate data by creating duplicate files, ONLY when there is enough free space (i.e., >=2X files and >=2Y blocks). If the number of free files is <2X and/or the number of free blocks is <2Y, but there are more than X files and/or Y blocks, then no message needs to be displayed. If the number of free files is <X and/or the number of free blocks is <Y a message similar to the following must appear:

   For example:

   “There is not enough space on the Memory Card in Slot <A or B>. <Title> requires <X> files and <Y> blocks to save. [Retry] [Continue without save] [Memory Card Screen]”

3. The third method is similar to the second method, but it resolves the safety issue because it requires that there be enough free space to create a separate file. Unlike the second method, it will not directly write to the original file when there is not enough free space. Thus, for an application that creates a 4-block file, the Memory Card needs to have 8 blocks of free space. This method may seem to be similar to duplicating, but it has merit in that you only need empty capacity of one file's worth when creating more than one file. The drawback to this method is that it cannot be used for files that are larger than one-half the Memory Card size. This method may confuse users, because it requires extra Memory Card space. So please display the following messages when there is not enough space to save.

   With this method there are 2 cases to consider.

   CASE A: Creating a file. If there are <X files and/or <Y blocks, a message similar to the following must appear.

   For example:

   “There is not enough space on the Memory Card in Slot <A or B>. <Title> requires <X> files and <Y> blocks to create a file. [Retry] [Continue without save] [Memory Card Screen]”

   CASE B: Overwriting a file. If there are <2X files and/or <2Y blocks, a message similar to the following must appear.

   For example:

   “There is not enough space on the Memory Card in Slot <A or B>. <Title> requires <2X> files and <2Y> blocks to overwrite a file. [Retry] [Continue without save] [Memory Card Screen]”

All three of the above methods have their advantages, so use the one that best fits the application.

If data has been destroyed and cannot be recovered, it will have to be erased and initialized. Display a message like the following, “The saved data is corrupted”. If you do not display a message before deleting and initializing the data, you may confuse the user.

3.10 The Reset process

The Nintendo GameCube™ RESET Button is not a system reset switch. When the user presses this button, the software is notified of that act and executes the reset process. There are many different ways for the application to perform the hot reset process, but if you are going to call the hardware reset function `OSResetSystem`, you must take note of the point explained below.

If a hot reset is called while writing to the Memory Card, the write operation will be interrupted and the file will be destroyed. For this reason, the function `OSResetSystem` specification is designed to wait until after the CARD API has ended before the hardware reset is performed. However, the `OSResetSystem` function simply waits for each API process to end. If, for example, the game attempts to write several files in sequence, there is a possibility that it will succeed in writing the first files but fail after reaching a certain point. Therefore, take special note of the fact that the application side must wait for the Memory Card to complete a process that involves a series of actions.

For more information about reset methods and procedures, see the " Nintendo GameCube™ Reset Guidelines."

3.11 Using multiple Memory Cards

Be sure to collect all saved data together in a single Memory Card. Users will become confused if the saved data is divided out among multiple Memory Cards. If the situation arises where a Memory Card does not have sufficient space to hold the saved data, then the use of more than one Memory Card is acceptable. However, please consider player convenience.

3.12 Changing the file size

In principle, do not change the file size (the number of blocks) once a file has been created. There is a risk that increasing the file size during the course of a game will lead to a situation where there is insufficient space and the player will not be able to advance in the game. Instead, create the file with sufficient memory space in the first place.

Please contact us at support@noa.com if you require a variable file size in order to support your application.

4. Large-Capacity Memory Cards

4.1 Overview

The CARD API and the Memory Card menu both incorporate support for future large-capacity Memory Cards (maximum 128Mbits). There is a limit to the number of files that can be created on a large-capacity Memory Card. The maximum number is 127 files. Once the card has 127 files, a new file cannot be created even if a free block is available.

With these large-capacity Memory Cards, it is possible to imagine cards with different block sizes (8Kbyte, 16Kbyte, 32Kbyte, 64Kbyte, 128Kbyte). For this reason, the Nintendo GameCube™ Memory Card Screen was designed so that files of differing sizes cannot be copied or moved between Memory Cards with different block sizes.

4.2 Cautions for Memory Cards with large-capacity

When creating applications, be sure to design them so they will operate normally even when Memory Cards of different capacity are inserted into the Nintendo GameCube™. If the application is created assuming a fixed capacity of 59 blocks, then when a Memory Card with 200 or more blocks in inserted the program may halt or exhibit other glitches, like odd screen displays because there are not enough digits. Please be careful so that these kinds of problems do not arise.

At this time, please support only Memory Cards with 8 Kbyte block size. Always use the function `CARDProbeEx` or `CARDGetSectorSize` to check the block size. When the block size is other than 8Kbyte, show a message "Memory Card in Slot A (or B) is not supported."

4.3 Items to check when using the large-capacity Memory Card Emulator

The Large-capacity Memory Card Emulator has been provided to check functionality with large-capacity Memory Cards. When creating your application, always use the Large-capacity Memory Card Emulator to check functionality. The combination of capacities (in megabits) and block sizes (or sector sizes, in kilobytes) that need to be checked with the Large-capacity Memory Card Emulator are listed in the table below.

Capacity / Block Size	4 Mb (59 Blocks)	16 Mb (251 Blocks)	32 Mb (507 Blocks)
8 KB/No Errors	Check Needed (Support Required)	Check Needed (Support Required)	Check Needed (Support Required)
16 Kbyte/ID Only	No Check Necessary	No Check Necessary	Check Needed (Incompatibility Message Display)
8 Kbyte/Program Error	Check Needed	No Check Necessary	No Check Necessary
8 Kbyte/Erase Error	Check Needed	No Check Necessary	No Check Necessary
8 Kbyte/INTB Disable	Check Needed	No Check Necessary	No Check Necessary

Check to ensure that your game works with 8-Kbyte block sizes on 4-Mb, 16-Mb, and 32-Mb capacity Memory Cards. Check to ensure that an incompatibility message is displayed for 16-Kbyte block sizes on 32-Mb capacity Memory Cards. Although checks are required to ensure correct functionality with 16-Kbyte block sizes on 32-Mb capacity Memory Cards, this does not mean that there is any plan to release Memory Cards with this specification. Please note that this is simply one example of differences in block sizes on large-capacity Memory Cards.

Program errors (data write failure), erase errors (failure to erase before writing data), and Disable INTB (no response from Memory Card) may also occur on the Large-capacity Memory Card Emulator. If these types of errors occur, the CARD API will return the `CARD_RESULT_IOERROR` code. In this situation, you should display a special error message. For information on messages, see the sample program. At the time of this release, the sample program only displays in English. For Japanese error messages, refer to the Nintendo GameCube™ Memory Card Screen.

If you generate an error when writing to the Memory Card, the system area may become corrupted, and the `CARD_RESULT_BROKEN` error code may be returned.

5. File Attributes

The following file attributes are available in the Memory Card file system.

- Copy Disable attribute (Copying prohibited on Memory Card Screen)
- Move Disable attribute (Moving prohibited on Memory Card Screen)
- PUBLIC attribute (Reading allowed from other applications)

Each attribute can be set independently. By default, files have the following properties: Copy Enabled, Move Enabled, and PUBLIC.

Use the Copy Disable attribute in cases like where the ability to save and repeatedly play a part of the game will make the game much less interesting, or where the data itself has value but the ability to copy that data will lower its value.

Use the Move Disable attribute in cases like where the game can only be played on a special Memory Card and transfer of the data to a normal Memory Card could cause problems.

Turn the PUBLIC attribute off when you want to give a file higher security.

Please set the file attributes only in special cases like these.

6. Cautions

This section brings together all of the cautions mentioned thus far. Please be sure to adhere to these conventions when creating your applications.

6.1 Check for free space

When the Memory Card is not inserted into Nintendo GameCube™ or does not have enough free space to create a file of the needed size, notify the user by displaying an appropriate message before the start of game play. Avoid the situation where the user starts the game and is only informed that there is insufficient space when he or she later elects to save the game.

6.2 Consider card removal/insertion

Design the program so normal game play can continue if the Memory Card happens to be removed and inserted while the power is ON, or if a different Memory Card is inserted.

6.3 Indicate when accessing (read and write) Memory Card

A message or some other means is required to tell the player when the game is accessing (reading and/or writing to) the Memory Card. The data becomes invalid if the Memory Card is removed during access. For this reason, employ some means that lets the user know when the Memory Card is being accessed. For example, right after the user selects [Save] you could display a message reading "Ready to access. Do not touch the Memory Card or the POWER Button." and then wait for an input from the user. Also display a message during access that reads something like "Accessing. Do not touch the Memory Card or the POWER Button."

6.4 Support for corrupted or damaged Memory Card

Initiate appropriate action when the Memory Card has been corrupted or damaged, and will not support reading/writing. In this case, the return value of the CARDCheck or CARDCheckAsync function is CARD_RESULT_BROKEN.

First, ask the user if he/she desires to format the Memory Card with a message like the following: "The Memory Card in Slot <A or B> contains corrupted data and needs to be formatted. Do you want to format?". Format the Memory Card only if the user agrees to do so. If this does not correct the problem (format was unsuccessful), display a message like the following, "The Memory Card in Slot <A or B> is damaged and cannot be used."

6.5 Data verification

If the Memory Card is removed when reading or writing is being done, the reliability of the data cannot be verified with the hardware or the library. Therefore, make sure a verification is done on the application side using a checksum, CRC, or similar measure. If the data is corrupted and cannot be recovered, display a message saying something like, “The Memory Card in Slot <A or B> contains a corrupted file. This file will be deleted. Press A Button to continue.”. As a solution to this problem of data being destroyed, we recommend the following when writing to a file.

- Duplicate the data
- Create and write to a new file, then delete the original file

6.6 Checking the Memory Card Screen

Make sure that icons, banners, and comments fit completely in the Memory Card Screen.

Specifically, comments are limited to 32 bytes for each line. But if you use a great deal of characters with wide character widths, some of the characters may be outside the display border and will therefore not be displayed. So be sure this a situation does not occur.

6.7 Do not access Memory Card with unsupported software

DO NOT access a Memory Card with software that does not support the Memory Card.

6.8 Data compatibility within same program

Observe data compatibility between Memory Cards, even when a program has been revised after mass-production. The Memory Card is a portable medium which can be carried by the user and exchanged between users. Data saved on a Memory Card must be compatible with all versions of a particular game.

6.9 Caution about creating multiple files

When a single application creates multiple files, make sure each file can be used separately. Do not have a situation where the deletion of one file can make other files unusable. For example, in a game where a saved data file is created for each player, be careful that the deletion of one player's saved data does not render the other player's saved data files unusable.

6.10 Caution about storing programs and special data

Do not save a program that the CPU can execute directly. Please notify “support@noa.com” when saving intermediate code that has program characteristics even though the CPU does not execute it directly.

6.11 Flash memory lifespan

Avoid unnecessary writes to the Memory Card.

The flash memory built into the Memory Card has a lifespan of about 100,000 write cycles. As the flash memory deteriorates the write speed slows down. Therefore, avoid excessive saves such as after every second, or after every move within the game.

6.12 Flash memory access speed

Do not create programs that set fixed durations for access to flash memory.

Flash memory devices have individual differences, and the amount of time it takes to access the same amount of data can vary greatly from device to device. Moreover, performance deteriorates every time data is written to the flash device, so access speeds will slow down over time.

6.13 Support for large-capacity Memory Cards

Provide support so problems do not arise when using large-capacity Memory Cards.

There are plans to market a large-capacity Memory Card in the future, and there is a possibility that changes will be made to the capacity (total number of blocks), the block size (flash sector size) and the access speed of these large-capacity Memory Cards.

In particular, be sure that Memory Cards with different numbers of total blocks can be used normally.

Do not support Memory Cards with different-sized blocks.

Use function `CARDGetMemSize` to check the capacity of the Memory Card and use function `CARDGetSectorSize` to check the block size.

6.14 Memory Card formatting Issues

Process properly when a Memory Card formatted for one market is inserted into a Nintendo GameCube™ for another market.

First, ask a user, for example, "Memory Card in Slot A needs formatting. Format now?", and format it only if a user agrees. Display a message something like "This Memory Card cannot be used." when it still cannot be used after formatting.

6.15 Displaying the game name in the first line of comments

If you create multiple files for one game, make the game name text string that appears in the first line consistent.

6.16 Support for Memory Card Slots

Since there are two Memory Card Slots (Slot A and Slot B), We recommend your game operates regardless of the slot into which users insert the Memory Card.

If the application only supports one slot, you must display a message to the user so they know which slot to use for the Memory Card.

6.17 Memory Card Menus in the application

Memory Card menus within the application should only handle files used by the application. Even if your application includes a screen that displays several files at once, the screen should only display files used by that application. Do not create a Memory Card menu that can operate all files, like the IPL Memory Card Screen.

6.18 Do not change the file size

As a matter of principle, once a file has been created do not change the file size (the number of blocks).