New High-level Memory Card Library (LibMcrd.lib)

Overview…..

- What is a “Memory card”?
- What is LibMcrd?
- The old way...
- The new way...
- Standard operations with LibMcrd
- Submission to Third Party QA
- Caveats
- Q & A

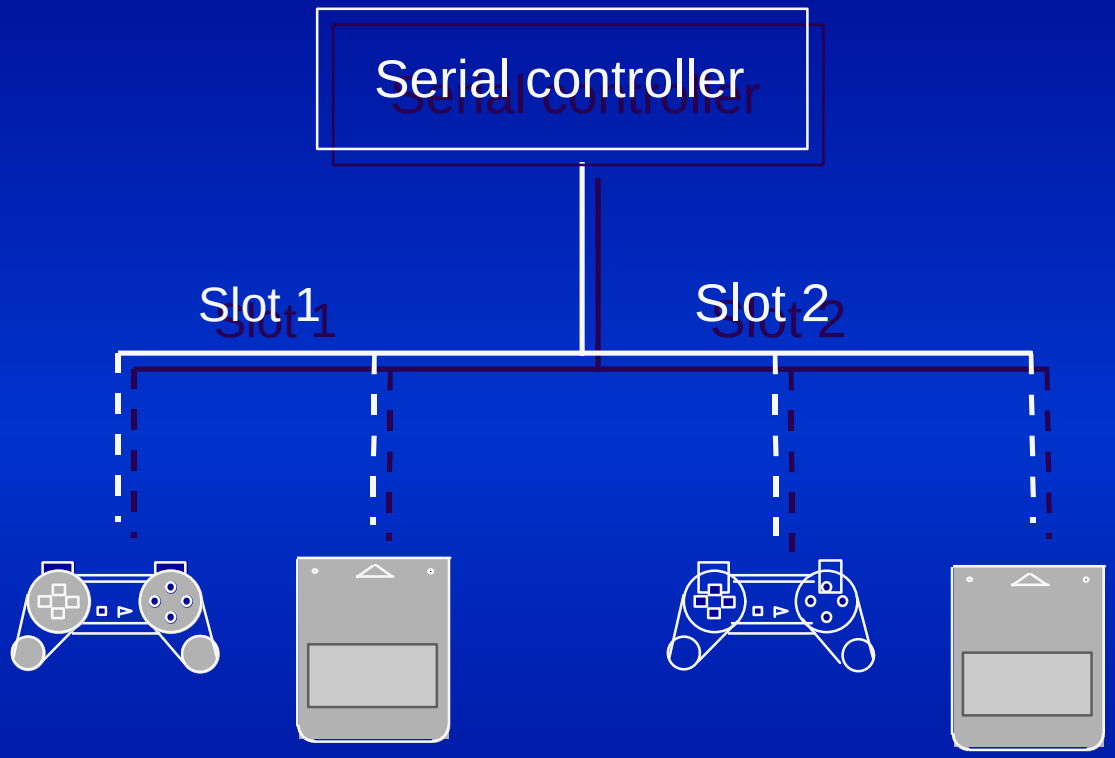
What is the Memory Card?

- The PlayStation Memory Card
  - Read / write memory
  - Removable
  - Shared
  - Used to
    - save scores, game position, player names, options etc.
    - as dongle for pre-release protection

What is a Memory card?

Specification

- Capacity:
  - 120 KB formatted
- Protocol:
  - Synchronous Serial communication
  - Shared with the controller
- Access:
  - Hardware - Max = 10 KB/sec
  - BIOS - 128 bytes / 2 Vsyncs
- Device Name:
  - "bu*XX*" Where *X* is the port number + card number.
  - Port number - A: 0x00 B: 0x10
  - Card number usually 0


Serial controller
Slot 1
Slot 2

Terminology

Sector

128 bytes. All read / write access are done in Sector units.

Block

8192 bytes. 64 Sectors. The minimum amount of space used by one file. 16 to a card, but only 15 available to applications. Often used to describe a sector.

Slot

File position. 15 to a card. Basically the same as a block.

Format

Same as formatting other storage devices. All new cards must be formatted. Can confirm via _card_load or MemCardAccept.

unconfirmed flag

Bit on Memory card which is set every time a card is inserted. Must be cleared by software using _card_clear or MemCardAccept.

Terminology (2)

File header

128 - 512 byte header that must be placed at the beginning of all Memory card files. Contains number of blocks, number of icons, icon data, etc.

Save File Name

The name of a a file as stored in the Memory card directory structure

Save Title

16 character (double-byte) Shift-JIS string at beginning the 5th byte of a of a Memory card file

What is LibMcrd?

- New to Version 4.0 of the PlayStation Libraries.
- Provides High-Level routines for easy Memory card handling.
- Automates many of the operations previously performed at application level using LibCard.
- Combines LibCard and file-handling routines in one easy-to-use function library.

What is LibMcrd?

- LibMcrd features:
  - Asynchronous status checking, reads, and writes
  - Automatic event set-up and handling
  - Automatic error Retry
  - File deletion
  - File creation
  - Directory entry Retrieval
  - Formatting

The Old Way, LibCard…

- Low-Level interface
  - Manual event setup and handling
  - Explicit initialization of file-system
  - Manual detection of card insertion and removal
  - open(), read(), and write() are used for file access
    - must ensure 128 byte aligned reads and write
    - must do bit Shifting to create a multi-slot file
    - must prepend file name with device name
  - Must use firstfile / nextfile to obtain directory entries

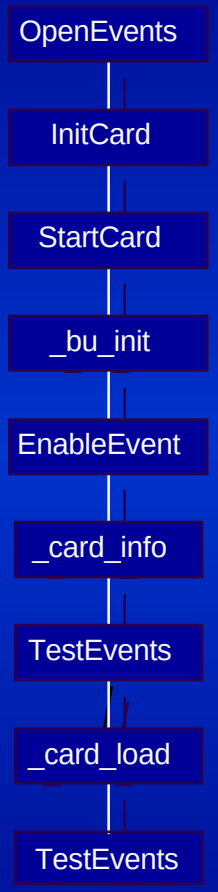
The New Way, LibMcrd...

LibMcrd High-Level Interface

- Memory card BIOS and file system are setup in one. function call.
- Events are setup and activated with one function call.
- For file access, device name no longer needs to be. prepended to file name.
- On creation file size is specified in block units.
- File Access not allowed with an unconfirmed card.
- One function call to retrieve directory entries.

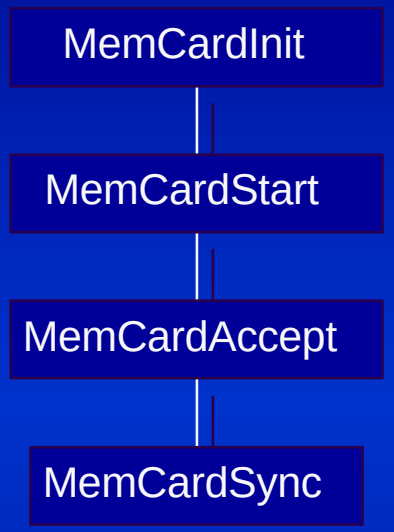
The Old Way….

- Fundamental Steps
  1) Install event handlers via OpenEvent()
  2) InitCard
  3) StartCard
  4) _bu_init
  5) EnableEvents
  6) _card_info
  7) Get result via TestEvent()
     if(NEWCARD) _card_clear;
  8) _card_load
  9) Get result via TestEvent()



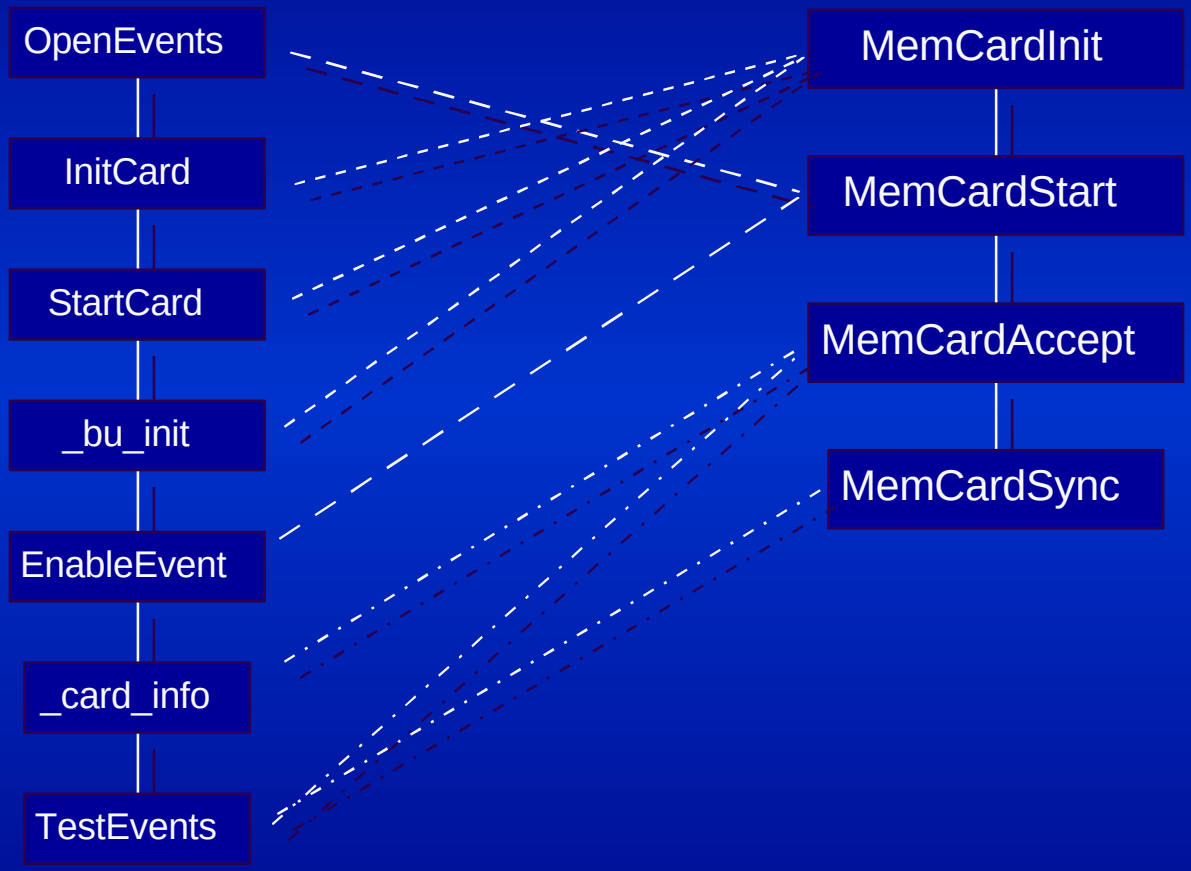
The New Way, LibMcrd...

- Fundamental Steps
  1) MemCardInit
  2) MemCardStart
  3) MemCardAccept
  4) MemCardSync



- 5 Steps less then LibCard!!!

General Program Flow, Old Vs New



LibMcrd functions

- MemCardAccept
- MemCardCallback
- MemCardClose
- MemCardCreateFile
- MemCardDeleteFile
- MemCardEnd
- MemCardExist
- MemCardFormat
- MemCardGetDirEntry
- MemCardOpen
- MemCardReadData
- MemCardReadFile
- MemCardStart
- MemCardStop
- MemCardSync
- MemCardWriteData
- MemCardWriteFile
- MemCardInit

LibMcrd function description - brief

- MemCardAccept ----------> Test Port/Card Status
- MemCardCallback ----------> Registers user-defined callback
- MemCardClose ----------> Closes last file opened
- MemCardCreateFile ----------> Creates a Memory card file
- MemCardDeleteFile ----------> Deletes an existing file from card
- MemCardEnd ----------> Shuts-down LibMcrd system
- MemCardExist ----------> Checks to see if a card is present
- MemCardFormat ----------> Formats Memory card
- MemCardGetDirEntry ----------> Retrieves directory info for file(s)
- MemCardOpen ----------> Opens memory card file
- MemCardReadData ----------> Reads blocks from file into memory
- MemCardReadFile ----------> Reads specified file into memory
- MemCardStart ----------> Activates LibMcrd system
- MemCardStop ----------> Halts LibMcrd system
- MemCardSync ----------> Gets sys status or blocks async func.
- MemCardWriteData ----------> Writes to individual blocks
- MemCardWriteFile ----------> Writes to specified file to card
- MemCardInit ----------> Initializes LibMcrd system

Asynchronous Operations

The following functions are asynchronous:

- MemCardExist
- MemCardAccept
- MemCardReadData
- MemCardReadFile
- MemCardWriteData
- MemCardWriteFile

When called the related operation is registered in a LibMcrd system structure and the registration status is returned.

1 : operation registered successfully

0 : registration failed

Registered operations are executed at VSync

Asynchronous Operations

- Only one asynchronous operation may be registered at a time.
- Attempting to register multiple asynchronous operations will result in the following message being printed and an immediate return with a value of 0 :

```
"Access Denied. : event multiple open\n"
```

- Upon successful registration of an Asynchronous function, the result of the previous operation is cleared.
- Therefore if registration fails you may want to retrieve the result of last operation before registering another .

Asynchronous Operations

- LibMcrd provides two ways to determine the results of asynchronous operations
  - Polling
    - MemCardSync
  - Callback
    - Register exit callback via MemCardCallback

Asynchronous operations

Macros for function types:

```
#define     McFuncExist         (1)
#define     McFuncAccept        (2)
#define     McFuncReadFile      (3)
#define     McFuncWriteFile     (4)
#define     McFuncReadData      (5)
#define     McFuncWriteData     (6)
```

Asynchronous Operations

Macros for operation results:

```
#define     McErrNone           (0)
#define     McErrCardNotExist   (1)
#define     McErrCardInvalid    (2)
#define     McErrNewCard        (3)
#define     McErrNotFormat      (4)
#define     McErrFileNotExist   (5)
#define     McErrAlreadyExist   (6)
#define     McErrBlockFull      (7)
#define     McErrExtend         (0x8000)
```

Polling

long MemCardSync(long *mode*, u_long **cmds*, u_long **result*)

- Arguments
  - *mode*
    - 0 : Wait. Block until completion of asynchronous operation
    - 1 : No Wait. Get status an return.
  - *cmds*
    - Address at which to store ID of last async operation.
  - *result*
    - Address at which to store result of last operation.
- Return Value
    - 0 : still active
    - 1 : terminated
    - -1 : No process registered

Callback

MemCB MemCardCallback(MemCB func)

- callback format:
  - *func_name*(unsigned long *cmds*, unsigned long *result*)
    - *cmds*
      - Macro indicating the last asyncrhonous function completed.
    - *result*
      - The result of the function specified in*cmds*.
  - typedef void (*MemCB)(unsigned long cmds, unsigned long rslt);
- Asynchronous
- Return Value
  - the address of the previously registered callback

Initializing/Activating LibMcrd

- Initialization is performed with:
  - MemCardInit
- Activation is performed with:
  - MemCardAccept

Initializing LibMcrd

void MemCardInit(long *val*)

- Initializes LibMcrd system.
  - *val* specifies wether or not access shared with controller
    - 0 : Not shared
    - 1 : Shared
  - Synchronous
  - Internally:
    - InitCARD(*val)*
    - StartCARD()
    - _bu_init

Checking Status of the Memory Card

```
/*Check card status, and clear card */
MemCardAccept(chan);

/* Get te results of MemCardAccept */
MemCardSync(0,0,&rslt);

/* If result is an error other than NewCard */
if(rslt!=McErrNone && rslt!=McErrNewCard) {
        ...
        ...
}
```

Initializing LibMcrd

- void MemCardInit(long *val*)
  - *val* should be 0 when using LibTap or LibGun
  - Should be called after InitPAD, InitGUN, InitTAP
  - If *val* is 1 it is not necassry to call StartPAD
  - Only call once

Activating LibMcrd

void MemCardStart(void)

- After initializing the memory card system with MemCardInit this activates it.
  - Synchronous
  - Internally
    - EnterCriticalSection
      - Opens and Enables 4 HwCARD events and 4 SwCARD events
    - ExitCriticalSection
    - VSyncCallback(_call_back_handler)

Initializing/Activating the Memory Card

```
ResetCallback();
SetDispMask(0);
ResetGraph(0);
SetGraphDebug(0);
InitPAD(0);

/*Initialize Memory card*/
MemCardInit(1);
/*Activate Memory card*/
MemCardStart();
```

Writing to the Memory card

- The following functions are used to write data from main memory to the card
  - MemCardWriteData
  - MemCardWriteFile

Writing to the Memory card

long MemCardWriteData(u_long * *adrs*,long *offset*,long *byte*)

- Writes data at *adrs* to Memory card file previously opened with MemCardOpen().
  - Must specify *byte* in 128 byte units else function will return with 0.
  - Asyncrhonous
    - 1 Vsync + 130 Vsyncs per block
  - Internally
    - calls lseek, write()
    - retries 16 times
    -

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	No Card
0x02	McErrCardInvalid	Comm. error
0x03	McErrNewCard	New Card

Writing to the Memory card

long MemCardWriteFile(long *chan*, char **file*, u_long **adrs*, long *offset*, long *bytes*)

- Writes data at *adrs* to the Memory card file specified by *file*.
  - *file* must exist.
  - Must specify *byte* in 128 byte units else function will return with 0.
  - Asyncrhonous
    - 1 VSync + 130 VSyncs per block
  - Internally:
    - calls open, lseek, write
    - retries 16 times

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	No Card
0x02	McErrCardInvalid	Comm. error
0x03	McErrNewCard	New Card
0x05	McErrFileNotExist	File not found

Writing to the Memory card

- Prior to write:
  - confirm card status
  - determine free space
  - maybe calculate checksum
- Following write:
  - Check result and retry on error
  - Should do checksum test

Reading from the Memory card

- The following function are used to read data from the card into main memory:
  - MemCardReadData
  - MemCardReadFile

Reading from the Memory card

long MemCardReadData(u_long **adrs*, long *offset*, long *bytes*)

- Reads data to *adrs* from the Memory card file previously opened with MemCardOpen().
  - Must specify *byte* in 128 byte units else function will return with 0.
  - Asyncrhonous
    - 1 Vsync + 130 Vsyncs per block
  - Internally
    - calls lseek, read()
    - retries 16 times

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	No Card
0x02	McErrCardInvalid	Comm. error
0x03	McErrNewCard	New Card

Reading from the Memory card

long MemCardReadFile(long *chan*, u_long **adrs*, long *offset*, long *bytes*)

- Reads data to *adrs* from Memory card file specified with *file*
  - Must specify *byte* in 128 byte units else function will return with 0.
  - Asyncrhonous
    - 1 Vsync + 130 Vsyncs per block
  - Internally
    - calls lseek, read()
    - retries 16 times

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	No Card
0x02	McErrCardInvalid	Comm. error
0x03	McErrNewCard	New Card
0x05	McErrFileNotExist	File not found

Reading From the Card

```
if(MemCardReadData((long *)&cardd->head[i],0,HEAD_SIZE) == 0)
  {
                printf("\nMemCardReadData : not registered");
                return -1;
  }

  MemCardSync(WAIT,&cmds,&result);
  switch(cmds)
  {
      case McFuncReadData:
      {
                switch(result)
                {
                        case McErrNone:
                        break;

                }
      }break;
```

Submission to Third-Party QA...

- Most common reasons for failing submission
  - Incorrect save file name
  - Failing to default to “No” when asking for format confirmation
  - ASCII characters in the file name

Submission to Third-Party QA

Save Title

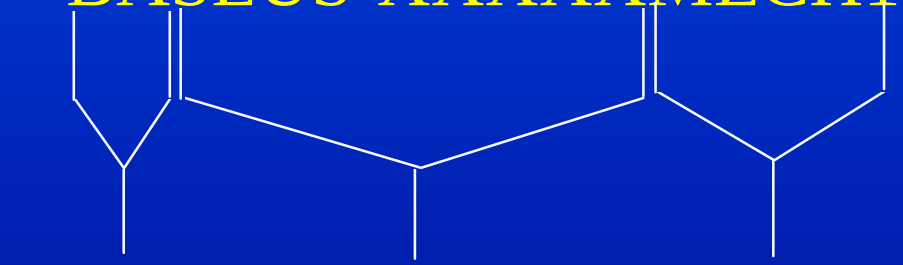
- The Save Title stored at the 5th byte of the file header must be in all Shift-JIS format. No ASCII characters. To do this:
  - Use function provided in sample program *asc2sjis* to convert at run-time or if practical convert before and hard-code.
    - remember to swap high and low bytes of Shift-JIS character
  - Verify Title using the Memory Card Manager program.
  - Save Title should not exceed 16 characters, for proper display on OSD.

Submission to Third-Party QA

Save File Name

"Each Title receives a unique file name that is determined by the region the game was developed for mixed with the product code number"

BASLUS-XXXXXMECH1



Region Code (US/SCEA)

Product Code

Save name (optional)

Information on other regions can be found in TRC

Submission to Third-Party QA

Formatting

When the program encounters an unformatted card the following steps must be performed:

- Clearly state that the card is unformatted and ask if the user wishes to format the card.
- Display “Yes” or “No”. **With “No” as the default selection**.
  - The button used to confirm a format should differ from the button that brought the user to the format prompt.
- Format Card
- Save Game
- Inform user of Save status.

Submission to Third-Party QA

Formatting

- Do not allow the user to explicitly format a memory card.
- In general, do all you can to avoid an accidental format

Caveats

- InitCARD(int val)
  - In 3.5 and earlier this must be executed in a critical section
    - If an interrunpt occurs during InitPad it may fail/crash
    - In 3.6 and up this has been prevented internally by entering critical section
  - Call after InitPAD
  - val should be 0 when used with Libtap or Libgun

- After creating a file with open() close it before accessing with read() or write()

Caveats

- long _card_chan(void)
  - In Lib 3.5 and 3.6 it is prototyped in kernel.h as void _card_chan(void), erroneously
- format() will always return 1 regardless of success or failure
  - Check for card presence and status prior to calling
  - confirm format afterwards by checking first 2 bytes of first block for “MC”

Caveats

- Proper initialization order
  - ResetCallbacks
  - ResetGraph
  - Initialize the pad
  - Initialize the card
  - Start card
  - Open events
  - Enable event
- Proper shutdown order
  - Close events
  - Stop card
  - Stop pad
  - Stop callbacks

Resources

- Documentation, FAQs, and Technotes
  - Technical Reference CD 2.0
    - ***\technote\memcard.doc***
      - initialization procedures
    - ***\technote\ShiftJIS.doc***
      - memory card save titles and shift-jis
    - ***\technote\techchck.doc***
      - pre-submission checklist
    - ***\training\winter95\advanced\a_mcard.pdf***
      - misc. memory card
    - ***\faq\sio9.pdf***
      - misc. memory card Q&A
    - ***\conf\scee\memcard.pdf***
    - ***\devref\Libref40.pdf***
    - ***\devref\libovr40.pdf***

Resources(2)

- Source Code
  - Programmers Tools CD 2.0
    - ***psx\sample\kanji\asc2sJIS***
      - example of converting ascii to Shift-jiis
    - ***\psx\samples\etc\card***
      - Example of how to make a Memory card file w/header. Icons, Title, etc.
    - ***\psx\samples\etc\mcrd***
      - various examples of using LibMcrd
    - ***\psx\samples\etc\old\cman***
    - ***\psx\samples\etc\old\card***

Resources(3)

- Books
  - ***Understanding Japanese Information Processing***
    - Ken Lunde
    - O'reily & Associates
- Developer Support
  - ***WebSite: www.scea.sony.com***
  - ***E-Mail: DevTech_Support@interactive.sony.com***
  - ***Hotline: (415)655-8181***
    - Like the Bat Phone of Support, reserve for urgent issues : -)

The End