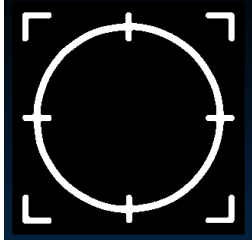
Memory Cards and Controllers

David Coombes
SCEE

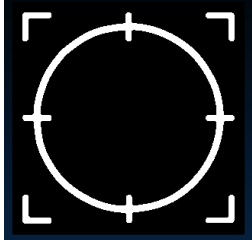
™



Introduction(1)

- Why are your here listening to me?
- Memory Cards
- Controllers
- Conclusion
- Brief Q.A. session

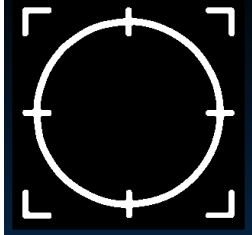
PlayStation™



Some words of wisdom (1)

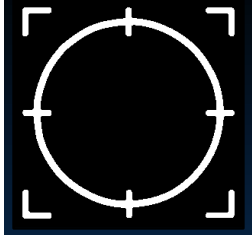
- Firstly...
  - before, any design work...
  - before any programming...
  - to save effort now and later...

PlayStation™



Some words of wisdom(2)

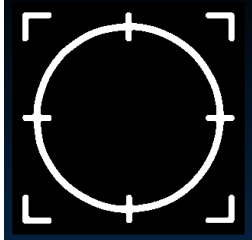
- Obtain, Read and Understand the Q.A. guidelines for each of the territories you want to publish in.
  - Japan is typically the hardest territory to get published in from a European perspective.
    - different video format
    - different typeface/additional characters
    - slightly different QA standards
  - Save yourselves time and money!



SIO System Overview

- Sio transfers (controllers, memory cards)
- All data transfer carried out inside the VSync
- Link Cable is handled differently
- Data transfer must be completed inside the VSync

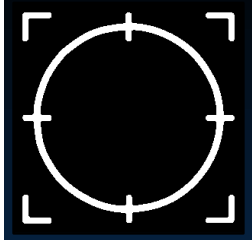
PlayStation™



Memory Cards

- Good
  - Solid State
  - Cool Technology
  - Inexpensive (No need for a complex expensive disk drive)
  - Very simple to use.

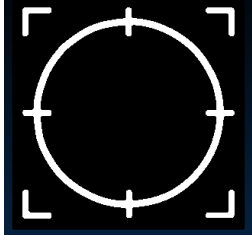
PlayStation™



Memory Cards

- Bad
  - Slow
  - Reputation for being hard to program
  - How many have been sold ? Well 60% actually

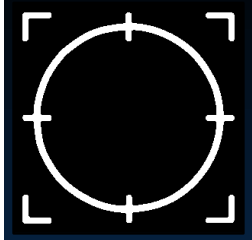
PlayStation™



Access Time

- Speed/Performance
- 4k per second (NTSC) 3.2k per second (PAL)
- Continuous read or write
- CPU load 3.2% (NTSC) 2.7% (PAL)
- Very Slow

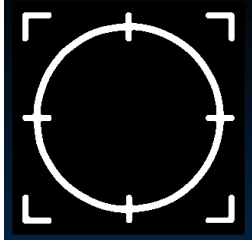
PlayStation™



Capacity

- 120K divided into 15 8K slots.
- Max number of files on one card is 15
- File size must be a multiple of 8K

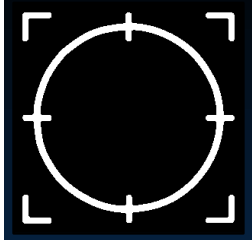
PlayStation™



Memory Cards

- File Header
- The header is used by the OSD Card Manager
- It must be correct or else your file will cause problems in this program and may result in data damage either to your file or another applications files.

PlayStation™



Memory Card File Header

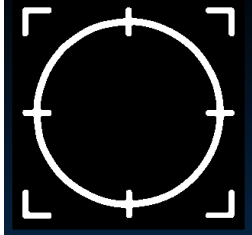
File Header

```
typedef{
    char Magic[2];          //This is always "SC"
    char Type;              //Number of icons 0x11=1   0x12=2
    0x12=3
    char BlockEntry;        //Number of 8K blocks used
    char Title[64];         //Must be in SJIS (32 characters)
    char reserve[28];       //Leave this alone
    char Clut[32];          //4 bit clut for icons (16*16bits)
    char Icon[3][128];      //3 icons   (16*16*4bits)
}_CARD
```


PlayStation™

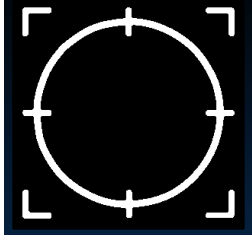
Memory Card File Header

- Title Field (Special Note)
- Full-size: Shift-JIS code only, 32 characters (64 bytes)
- If the number of characters is less 32, the character string must be finished with a null character (0x00), or unused space within the 32 characters must be padded with blanks(0x2020).
- The use of ASCII codes is prohibited.



Memory Card Icons

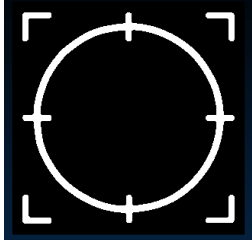
- How do I create my Icons?
  - Incbin a 16*16 4 bit tim file
  - Copy the clut and pixel data across into the correct fields.



Special Note on Icons

- Due to a bug in the OSD Memory Card program
  - If your game only uses :1 slot.
    - (only have 1 icon)
  - :2 slots
    - (have 1 or 2 icons)
  - :3 or more slots
    - (have 1 or 2 or 3 icons)
- This only applies to the PAL systems

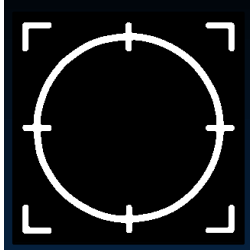
PlayStation™



Initialising Memory Cards

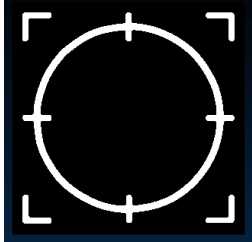
- Install memory card system at start of program and do not remove.
- InitCARD does not like to be interrupted.

PlayStation™



Detecting the Presence of Memory Cards

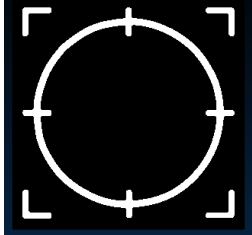
- Use services provided by the BIOS
- Use _card_info function
- Interpret the results



Abnormal Processing

- Removal of card during operation / power failure
- PlayStation™ OS provides no user interface for abnormal processing.
- Your application must provide a suitable interface.
- Most of what is required is common sense.

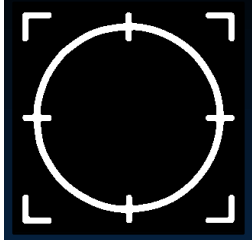
PlayStation™



Saving your game

- Required Dialogue
  - Is there a card present?
  - Is it formatted?
  - Is there space?
- Appropriate processing for each (see flow chart in notes)

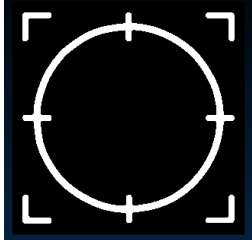
PlayStation™



Accessing memory cards

- Via device driver
  - device name is bu: (backup unit + channel number)
  - Port 1 00 Multi Tap 00 01 02 03
  - Port 2 10 Multi Tap 00 11 12 13

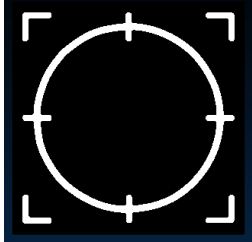
PlayStation™



Accessing file structure

- Use the firstfile/nextfile functions to get a list of the files and their sizes.

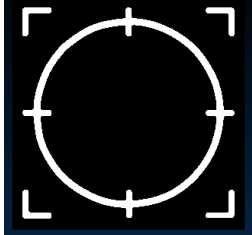
PlayStation™



Deleting a memory card file

Example

```
// Delete the file DevConDemo in port 0
//*****************************
delete("bu00:DevConDemo");
```



Creating a memory card file

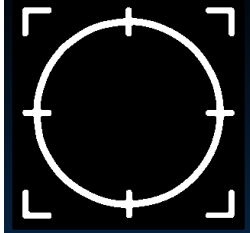
Example

```
// Create a file DevConDemo in port 0 with a
// block size of 3 (24Kb)
//*************************************
fd=open("bu00:DevConDemo",
                O_CREAT|(3<<16));
close(fd);  // don't forget this !
```


PlayStation™

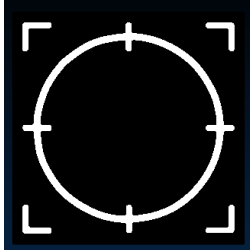
Important notes on file creation (1)

- The filename is 21 ASCII characters, however...
  - File name is defined by area code i.e. BE for Europe
  - 10 digit product code i.e.. SLES_00069
  - It should be terminated with NULL (0x00)
- The remaining characters may be used for personalisation or to support multiple versions.
  - i.e.. BESLES_00069DONUT123(0x00)



Important notes on file creation (2)

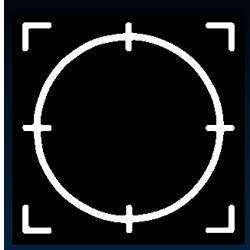
- Once a file is created its size may not be changed, without deleting and recreating the file. Make sure the file is big enough to start with.
- After creating a file it must be closed before it can be written to.



Writing to a memory card file (Blocking)

- Can only write to a file that already exists
- Must write a multiple of 128 bytes
- Must not write beyond the end of the file

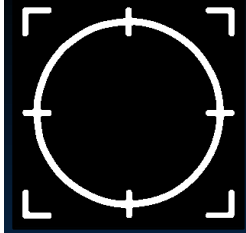
```
fd=open("bu00:DevConDemo",O_WRONLY)
   ;
i = write(fd,buff,128);
```



Reading from a memory card file (Blocking)

- Open file in Read only mode
- Must read a multiple of 128 bytes

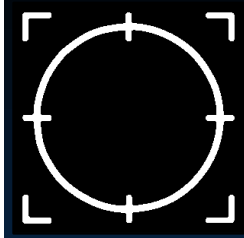
```
fd=open("bu00:DevConDemo",O_RDONLY);
i = read(fd,buff,128);
```



Writing to a memory card (Non-Blocking)

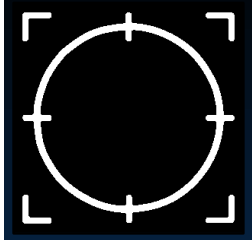
- Access speed is dependent on the amount written per iteration
- This also affects the frame rate
- Find the right balance for your needs

```
fd=open("bu00:L01",O_WRONLY|O_NOWAIT);
while((ret = write(fd,data,384))!=0);
```



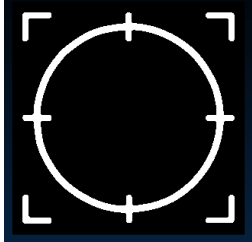
Reading from a memory card (Non-Blocking)

- Access speed is dependent on the amount read per iteration
- Note: file pointer update in the read function is bugged
- Use lseek to correct!
- fd=open("bu00:L01",O_RDONLY|O_NOWAIT);
- while((ret = write(fd,data,384))!=0);



When accessing the card

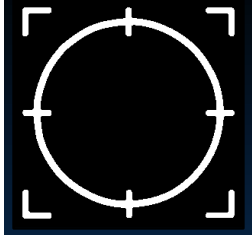
- When reading or writing to the card.....
- Don't omit operation failure checks, for example, check for card removal during a read or write.
- Compress your data to reduce access time
- Consider using checksum for each 128 bytes written and save all data at least twice to back data up



Formatting Cards

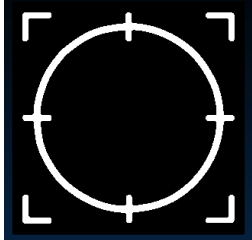
- New Cards are supplied unformatted.
- 2 ways of formatting a card.
- Use the _auto_format function. You should not do this inside a final product. This is for testing only.
- Use the format function i.e. format(bu00:)
  - note format is buggy. It always returns a 1, this is no guarantee that the format succeeded.

PlayStation™



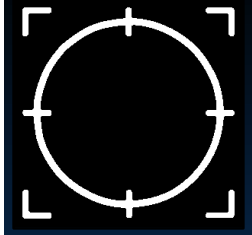
Checking format

- Use _card_load() and then test for events again (which is slow) or.....
- Read the first two bytes on the card, if they are “MC” then the card is formatted



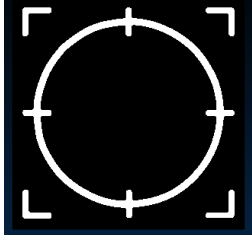
Limitations

- Speed of access, the less data you store on the cards the less the game flow will be disrupted by reading/writing to the card
- Cannot play SPU sound whilst accessing a card because both occur inside the VSync and must finish inside the VSync, a time-out will occur.
- This may not be true?



Controllers

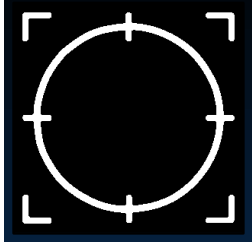
- As important as graphics and sound
- The only input to the game
- One device must be suitable for controlling
  - cars, boats, planes, helicopters
  - football teams, robot rabbits, an angry man in a dress....
- Its got to feel right for all these tasks



Controllers

- Standard Controller
- Mouse
- NegCon
- Analogue Device (Joystick)
- Light Gun (Not currently supported)
- Multi Tap (support up to 8 controllers on a single PS)

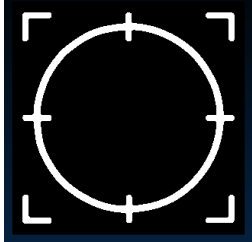
PlayStation™



Standard Controller

- Standard Controller by Sony
- Loads of clones, some approved, some not
- 12 Digital buttons!
- 3d Control system!

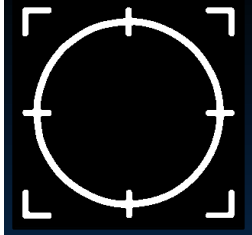
PlayStation™



Mouse

- Classic two button design from Sony
- Shame there are not more buttons!

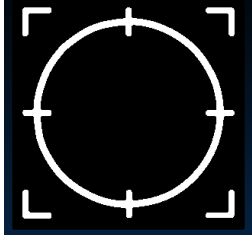
PlayStation™



NegCon

- Compact Analog device from Namco
- 4 analog channels
- directional button and 3 digital buttons
- Good for driving games.
- Compatible Steering wheel and pedals from MadCatz

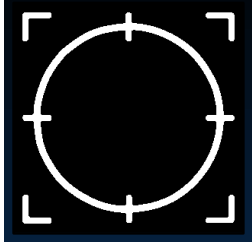
PlayStation™



Analog device

- Big Analog device from Sony
- 4 analog channels
- lots of buttons
- Good for flight sims/ helicopter games /tank games

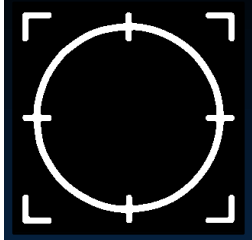
PlayStation™



Gun

- Konami device
- Trigger plus two buttons
- It does get heavy quickly
- No library support at the present

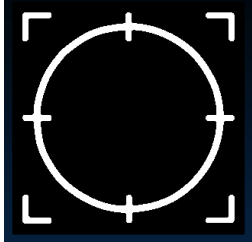
PlayStation™



Multi Tap

- Allows four Controllers to be connected to a single port
- Allows up to 8 players simultaneously.
- Allows connection of 8 memory cards.

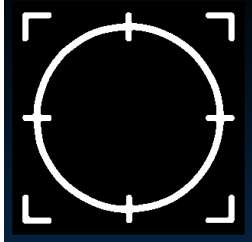
PlayStation™



The Xfiles

- Namco paddle
- Guitar Pick
- Skateboard simulator
- V.R. headset
- Rowing machines (custom applications)

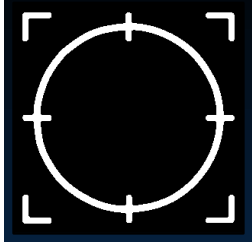
PlayStation™



Controllers

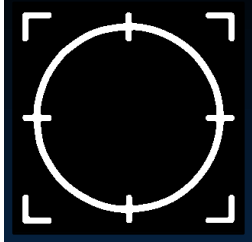
- QA demands that your product:
  - Detect the absence of the correct controller class and respond accordingly
  - Be fully playable with the standard controller
- See the Q.A. Guidelines Document on the BBS for complete details of what is required for Europe

PlayStation™



Initialising Controllers (1)

- **Do not use:**
  - PadInit(); (This is from libetc)
  - PadRead();
  - (these functions are used by all of the sample code)



Initialising Controllers (2)

- **Do use:**

  InitPAD(); (This is from libapi)

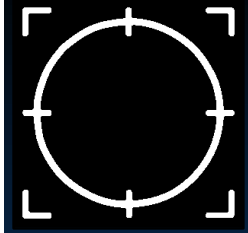
  StartPAD();

- or

  InitTAP(); //when using the multi tap.

  StartTAP(); //use libtap.lib.

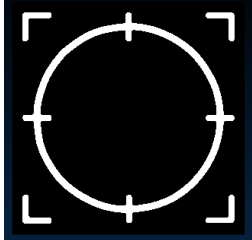
  - This applies to library all versions up to and including 3.4



Initialising Memory Cards and Controllers (3)

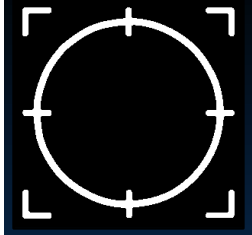
- Initialise controllers before cards
- See example in notes

PlayStation™



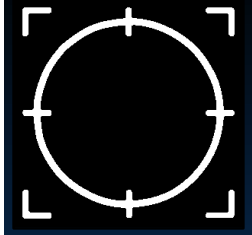
Explanation

- buffer1 and buffer2 will be updated every VSync
- Why call ChangeClearPad(0) ?
  - Because StartPad and StartCard both call ChangeClearPad(1); which disables the VSync interrupt handler.
  - You need to turn it back on else your VSync callback functions will never be called



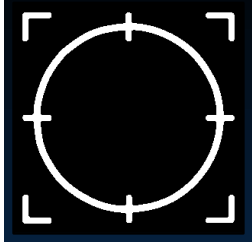
Controller Data Packets

- CTRLLER.H
- Provides a series of macros which allow simple smooth access to the controller data packets. You can use these in your game or base your own routines on these.
- This file is available from the SCEE BBS and is also on current SCEE release CD



Controllers

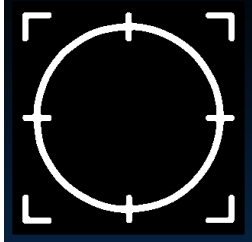
- Example of macro usage from ctrller.h
- if the data packet in buffer1 is good and its a standard controller connected then read the controller data.
- if the front left top "shoulder pad" is pressed call the map() function.



Conclusion

- Never underestimate the importance of a good control system
- Make your game easy and fun to play with the standard controller and another controller
- Memory cards give you the ability to increase the depth and complexity of your games.
- Save default settings\preferences.

PlayStation™



Therefore

- You should be able to make the best games ever
- So go for it!

PlayStation™