Applejack Input Device Driver

version 003

Abstract: The Applejack input device driver combines the features of a game-player pad and a mouse or trackball in a small handheld device. Generally, Applejack input devices are attached to a Pippin Power Player (a CD-ROM multimedia player device derived from the PowerPC Macintosh).

Please send questions and comments via e-mail to pippindev@apple.com.

1 Introduction

The Pippin input device, Applejack, is a custom ADB device containing trackball information and a series of 18 button states. The Applejack driver resides in the Pippin ROM and loads at system boot time. On a Macintosh, the Applejack driver resides in the *Applejack 2.2.0* system extension file. [1]

The Applejack driver uses a `pipp' button mapping resource which allows any Applejack button to be mapped with either a mouse button or keyboard key function. The trackball data, however, is always treated as a ``mouse'' and there is no provision for remapping it.

While provision for default button mapping is provided, applications can also include a custom `pipp' resource for setting button functions. On Pippin ROMs, a custom `pipp' resource would be loaded automatically when an application is launched. For applications that do not use mouse or keyboard mappings (not highly recommended), the Applejack raw data is directly readable from the driver.

1.1 Setting Up

The Applejack Software Developer's Kit diskette contains the *Applejack 2.2.0* application for editing the `pipp' mapping resource, and an *Applejack 2.2.0* system extension file.

1. From the SDK diskette, copy the application file to your Macintosh hard drive.
2. Drag-and-drop the *Applejack 2.2.0* system extension file to the ``Extensions'' folder in your System Folder.
3. Ensure that the Applejack input device driver(s) are plugged into the ADB jack adapter(s) on the backside of your Macintosh.
4. Restart the Macintosh to initialize the *Applejack 2.2.0* system extension file. [2]

You are now ready to customize the Applejack input device driver.

2 Customizing the Applejack

There are two ways to customize the `pipp' mapping resource of the Applejack input device driver(s).

The simplest way involves using the *Applejack 2.2.0* application as an interface for redefining device button mapping. This mechanism is as simple as aligning an attached Applejack device (or possibly, unattached Applejack device, as the case may be) with a visual device on the Macintosh screen. By clicking on the on-screen buttons, a developer customizes an Applejack device(s).

The second way involves modifying the `pipp' mapping resource code files manually, specifically defining how the button mechanisms on the device(s) should work.

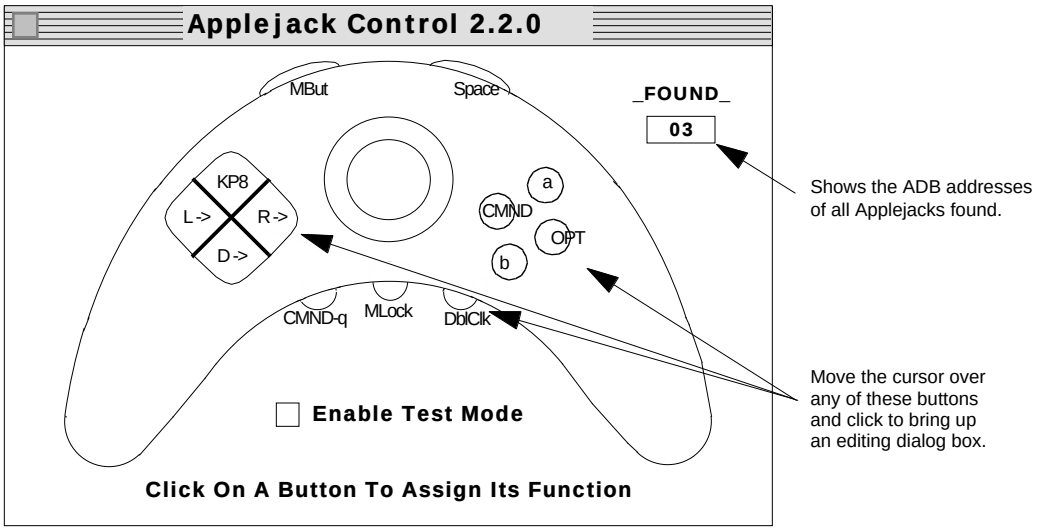
---

1. The *Applejack 2.2.0* system extension for Macintosh requires that the Cursor Device Manager be present. Any Power Macintosh or “AV” Macintosh has the Cursor Device Manager in ROM.
2. If the displayed *Applejack 2.2.0* system extension icon is crossed out while rebooting your Macintosh, your Macintosh either could not find an Applejack device plugged into the ADB jack, or the Macintosh you are using does not have the Cursor Device Manager in the ROM (i.e., you are not using a PowerMacintosh or “AV” Macintosh).

2.1 The Applejack Control 2.2.0 Application

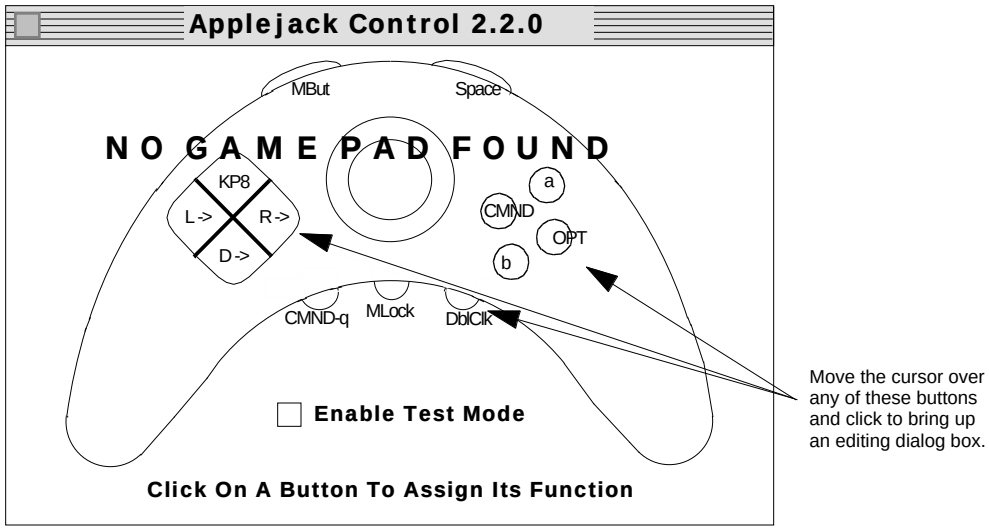
The *Applejack 2.2.0* application is the interface used for editing the `pipp' mapping resource. Double-click on the *Applejack 2.2.0* application icon and a window similar to Figure 1 will appear.

Figure 1 Applejack Control 2.2.0 Application Launch Window



If the *Applejack 2.2.0* application does not find any Applejack device(s) physically connected to the bus, the window displays a ``NO GAME PADS FOUND'' message across the window instead, as follows.

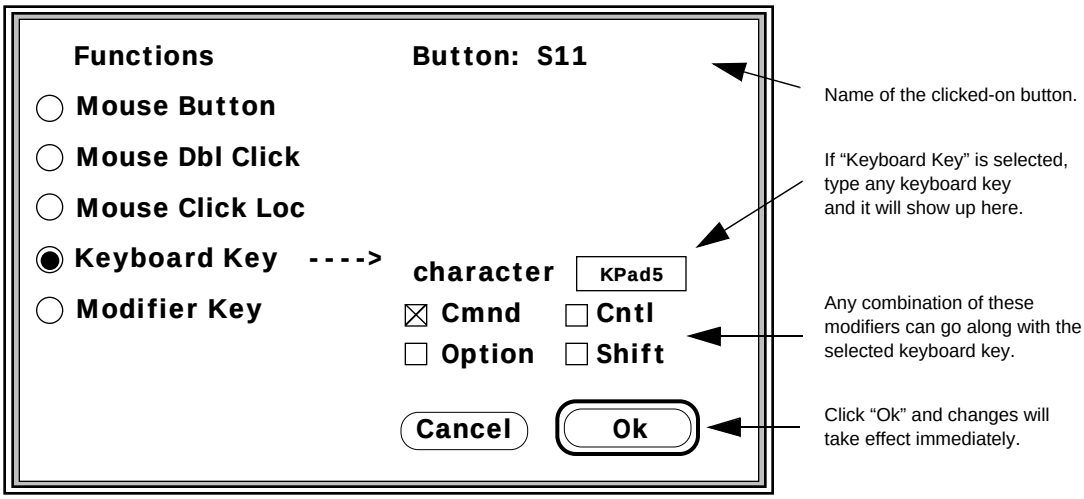
Figure 2 NO GAME PAD FOUND Window



Regardless of whether an Applejack device(s) is connected or not, the button settings may be edited. By clicking on desired buttons, a developer can customize the Applejack device(s).

A dialog box similar to Figure 3 or Figure 4 appears, depending on the button clicked on.

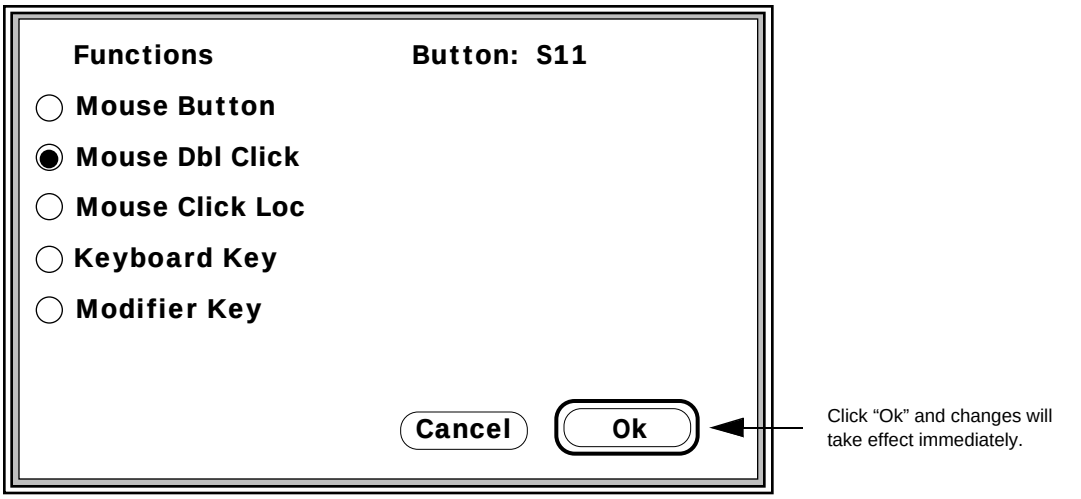
Figure 3 Applejack Control 2.2.0 Dialog Box--Customizing Keyboard Key Functions



When defining specific keyboard keys, select the “Keyboard Key” option, and press the key desired, which will appear in the character fill-in box.[3] Also, modifiers can be designated (click on appropriate box) with a specific keyboard key, if desired. Then, click on “Ok” to proceed.

When defining mouse button functions, click on the appropriate “mouse” functions, as desired, then click on ``Ok'' to proceed.

Figure 4 Applejack Control 2.2.0 Dialog Box--Customizing Mouse Button Functions



By clicking ``Ok'' after defining each button, the Applejack driver's local data is updated. If an Applejack device is attached, the newly defined button functions can be tested immediately with the newly ``customized'' Applejack device(s).

3. The KeyCaps desk accessory will not always draw the keys correctly for Applejack. Afterall. how does one define keys for an Applejack keyboard based on a standard typing keyboard?

By quitting the *Applejack 2.2.0* application, the `pipp' resource within the *Applejack 2.2.0* system extension file is updated. If you would like to write the `pipp' resource into a separate file, use the ``File:Make Resource..." menu item.

2.1.1 Applejack Default Button Settings

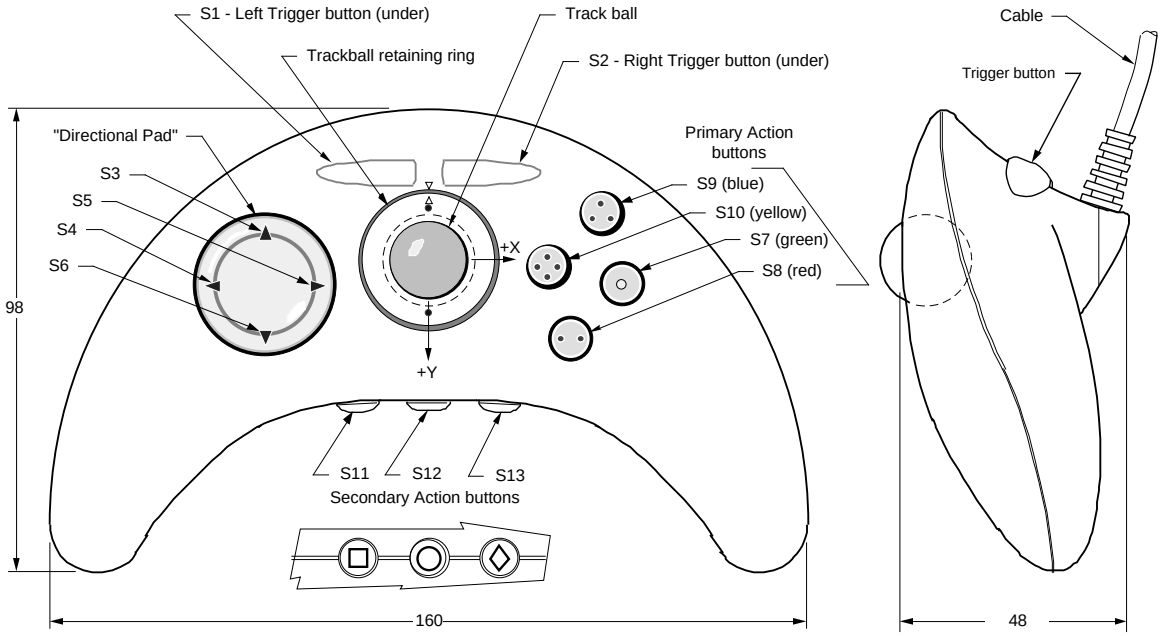
The Driver in the Pippin ROM creates default settings for the Applejack buttons if there is no Applejack extension installed. The default Applejack button settings are as follows:

Button	Description
Front left button	Extended Keyboard F11
Front middle button	Extended Keyboard F12
Front right button	Extended Keyboard F13
Blue button	Extended Keyboard F14
Yellow button	Letter J
Cross Up	UP Arrow
Cross Left	LEFT Arrow
Cross Right	RIGHT Arrow
Cross Down	DOWN Arrow
Red button	Extended Keyboard F8
Green Button	Letter K
Right Fire	Mouse Button
Left Fire	Mouse Button

2.2 Modifying the `pipp' Button Mapping Resource

Alternatively, `pipp' resources can be built and keys (functions) remapped by looking at the driver's global data. Figure 5 illustrates the Applejack Input Device Driver and its corresponding button mapping.

Figure 5 Applejack Input Device--Button Mapping



Notes:
1. Dimensions and shape are shown only approximately.
2. Switch and element reference designators shall correspond with circuit schematic and µC firmware designations.

The `pipp' resource is identical to the driver's global data and is a structure of type AJGlobalData. This global data is pointed to by the `.refCon` field of the Cursor Device Manager record for each Applejack device, as defined in the following code.

```
typedef struct SwitchData {
Byte    function;
Byte    modifiers;
Byte    keyCode;
Byte    charCode;
} SwitchData, *SwitchDataPtr;

typedef struct AJGlobalData {
long    signature;
Byte    MyTalkR0;
Byte    handlerID;
Byte    ourMBState;
Byte    ourLockState;
long    switchStates;
SwitchData      switcheMappingArray[18];
long    nextjADBProc;
long    cursorHandler;
} AJGlobalData, *AJGlobalDataPtr;
```

Most notably in the above code is the signature. The driver installer has initialized the signature to equal `pipp' so that applications can look through the ADB device table and find this structure. Further, the function field of the `SwitchData switchMappingArray` is defined as follows:

```
enum {
   kNothing       = 0,    // maps to nothing
   kMouse         = 1,    // maps to standard mouse button operation
   kMouseDblClick = 2,    // maps to mouse button double click
   kMouseLock     = 3,    // toggles the mouse button state
   kKeyboard      = 4,    // maps to a keyboard key
   kFrontPanel    = 5,    // not used
   kModifier      = 6     // maps to a modifier key
};
```

Each element of the `switchMappingArray` array represents one of the switches on Applejack. The array index is equal to the bit numbers, as shown in Figure 6.

Figure 6 ADB Register 0 Four-Byte Packet and Bit Number to Button Mapping

<— Bytes —>

	<——— Bits ———>							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	b17	y6	y5	y4	y3	y2	y1	y0
1	b16	x6	x5	x4	x3	x2	x1	x0
2	b15	b14	b13	b12	b11	b10	b9	b8
3	b7	b6	b5	b4	b3	b2	b1	b0

Bits y6 - y0 are the 7-bit, 2’s complement Y-axis displacement.

Bits x6 - x0 are the 7-bit, 2’s complement X-axis displacement.

Bit #	S #	Button Map
b17	S1	Left Fire
b16	S2	Right Fire
b15	S7	Green
b14	S8	Red
b13	S6	Down
b12	S5	Right
b11	S4	Left
b10	S3	Up
b9	S10	Yellow
b8	S9	Blue
b7	NC	
b6	NC	
b5	NC	
b4	NC	
b3	NC	
b2	S13	Right System
b1	S12	Middle System
b0	S11	Left System

If a button is mapped as `kKeyboard`, then the Applejack driver posts a `keyDown` event with the `.keyCode`, `.keyChar` and `.modifiers` fields in the event record.

The following code shows an enumeration of equates for the `.modifiers` field.

```
enum {
  kCommandBit    = 0,
  kShiftBit      = 1,
  kCapsLockBit   = 2,
  kOptionBit     = 3,
  kControlBit    = 4
};
```

If a button is mapped as `kModifier`, then the Applejack driver sets the `keyMap` global for the `keyCode` specified in the `.keyCode` field. The other fields are ignored in this case.

The following code is another enumeration of equates for the `.modifiers` field.

```
enum {
  kCommandKey             = 0x37,
  kShiftKey               = 0x38,
  kCapsLockKey            = 0x39,
  kOptionKey              = 0x3A,
  kControlKey             = 0x36
};
```

To change the mapping of a button, map a pointer to the `AJGlobalData` structure, then put in new values for the `.switches` fields for the desired switch. A good reason to do this would be if you wanted to read the Applejack raw data but did not want a button to also be generating system events.

The following code maps a pointer to the `AJGlobalData` structure for each connected Applejack, and then changes the selected switch.

```
void ChangeSwitchMapping (short whichSwitch, short function, short modifiers,
short keyCode, short charCode);

void ChangeSwitchMapping (short whichSwitch, short function, short modifiers,
short keyCode, short charCode);
{
ADBAddress       address;
ADBDataBlock     dataBlock;
AJGlobalDataPtr  myAJ;
CursorDevicePtr  myCrsrDev;
short            index,i;

//We need to copy these changes into the currently installed drivers.
//Where are the Applejacks?

//The Applejack driver uses the .refcon field of the cursor device record to
//store a pointer to its globals, a AJGlobalDataPtr.

if(whichSwitch>=0 && whichSwitch<=17)
  {                                          // where are the AppleJacks?
  index = CountADBs();
  while(index>0)
      {
      address=GetIndADB(&dataBlock,index);
      if((dataBlock.origADBAddr)==kDevAddr)
          {                                  // make sure that this is really us
          if(dataBlock.dbDataAreaAddr)
          {
```

```
            myCrsrDev=(CursorDevicePtr)dataBlock.dbDataAreaAddr;
            if(myCrsrDev)
                {
                myAJ=(AJGlobalDataPtr)myCrsrDev->refCon;
                if(myAJ->signature=='pipp')
                    {                       // got it.
                    myAJ->switchMappingArray.function[whichSwitch]=function;
                    myAJ->switchMappingArray.modifiers[whichSwitch]=modifiers;
                    myAJ->switchMappingArray.keyCode[whichSwitch]=keyCode;
                    myAJ->switchMappingArray.charCode[whichSwitch]=charCode;
                    }
                }
            }
        }
    index--;
    }
```

Some other ways to use this function could be as follows:

- Map the yellow button (S10, bit 9) to do nothing:

  ```
  ChangeSwitchMapping(9,kNothing,0,0,0);
  ```

- Map the red button (S8, bit 14) to be equal to command-Q (quit):

  ```
  ChangeSwitchMapping(14,kKeyboard,((1<<kCommandBit)),0x0C,'Q');
  ```

- Map the green button (S7, bit 15) to be equal to the shift key:

  ```
  ChangeSwitchMapping(15,kModifier,0,kShiftKey,0);
  ```

3 Reading Raw Applejack Data

To read raw Applejack data yourself, use code like the previous sample code, and ensure that you map a pointer to the `AJGlobalData` for each connected Applejack. You need to do this only once, but remember the pointer since it will not be moved or purged.

Then, examine `myAJ->switchStates` to read the state of a button. Each button correlates with a bit in `switchStates`. The bit number is the same as defined in the `.switcheMappingArray` array index.

For example, suppose you want to know if the yellow button is pressed; the following statement might apply:

```
if(!((myAJ>switchStates)&(1<<9)))   // 0==DOWN
  {                                 // 1==UP
  Do something useful here, yellow button is down...
  }
```

Be sure to keep separate pointers for each Applejack found (maximum of 4) even if you only support a single player play. If the bit number returns a value of ‘0’, then the button is pressed. If a value of ‘1’ is returned, the button is not pressed. No other event loop is required.