GUIDE DE L'UTILISATEUR EPIC

(Versions at the time of this writing EPICenter V1.0.5.2, EPICIO.DLL V1.7.0.0, EPICUSB.SYS 1.2.5.1)

Note: References to Game Controllers buttons are the actual button. Game Controllers buttons start with 1, actual buttons start with 0. So button 20 will show up as 21 in Game Controllers.

EPICISA used button numbers to link a physical button to an EPIC procedure.

Definebutton,defbtn were used to accomplish this. The button number was translated to Module,row, bit by reading the BUTTON.CFG translate file. You could figure the button number by executing TEST128.EXE (DOS) or TEST128W.EXE and watching which button changed when, for example, you changed your gear lever. You would use this number in the definebutton statement.

```
defbtn(120,on, GearUp)
defbtn(120,off, GearDown)

:GearUp {(nqw(BtnP,33)}
:GearDown       {(nqw(BtnP,34)}
```

the events 33 and 34 would correspond to buttons 1& 2 on Game Controllers, device 1. You figured this by dividing the event number by 32 for the device and the remainder was the button number.

This would correspond to section [JOYSTICK_01] in FS98/FS2000.CFG and would look like:

```
BUTTON_DOWN_EVENT_01=GEAR_UP
BUTTON_DOWN_EVENT_02=GEAR_DOWN
```

It would be nice to be able to use names for all of this to see what things meant, without having to calculate.

Also tech support always had to have the BUTTON.CFG file for each user if this file was changed from the shipped version.

#defines could be used for all of this, but that generates more typing and you still have figure everything out.

```
#define GEAR            120
#define GEAR_UP         33
#define GEAR_DOWN       34
```

Then this could be changed to:

```
defbtn(GEAR , on,GearUp)
defbtn(GEAR, off, GearDown)

:GearUp {nqw(BtnP,GEAR_UP)}
:GearDown       {nqw(BtnP,GEAR_DOWN)
```

Which is easier to read.

Connectors and Devices

With EPICenter and EPICUSB the concept of "connectors" and "devices" was introduced. Since this confused many users, I am writing this document.

A "device" is a collection of switches, buttons, displays, rotaries,analogs, etc. An autopilot panel would be an example of a device. Firewall controls would be an example of a device and could have light switches, engine starter, gear switch, flaps, led or lamp indicator, stall warning, etc. A device is defined using 1 connector that it would "plug" into. This does not have to be physical plug connection, and in reality be just the wires going to various analogs, and mod, rows.

See SYNTAXUSB for various device definitions (http://www.microcockpit.com/2009/pdf_bin/Syntax_USB_1082.pdf).

Devices elements can be used to pass events to Direct Input (which show up in Game Controllers), to pass events to programs communicating with EPICIO.DLL, or internally to execute procedures when a button changes state.

If a button is not attached to an internal procedure, it will generate an automatic event.

This event can go to Direct Input, the DLL or both. This is controlled by keywords SEND_BUTTON_DLL, SEND_ANALOG_DLL , SEND_POV_DLL (EPROM V5.2 and above) in the device definition.

#pragma hid_snd_rpt xx where xx is the number of devices to send to HID controls which devices will send automatic events to Direct Input. (EPROM V5.1 and above).

Recommended device order:

1. Devices to send to Direct Input and alternately to EPICIO.DLL
2. Devices to send to EPICIO.DLL (using options(SEND_BUTTON_DLL))
3. Devices with physical switches to be used to execute EPIC procedures, displays, outputs, etc.
4. mousedevice

Devices plug into "connectors" . Connectors are a collection of analogs and module rows to be scanned (checked by EPIC for changes). Different connectors can have the same analogs and module rows as other connectors, but the end definition(device level) of the actual switch (Mod,row,bit) MUST be unique. The same "input" analog (physical analog) can be used in multiple devices and be used multiple times in the same device, if needed.

Back to the Gear example:

If the gear switch is on Module 1 row 4 bit 3 for example.

The connector is defined (use all analogs and all rows (0-15) initially to simplify device definition.

```
connector(Mod1)
{
  analog(0);
  ..... other analog definitions 1 to 15.
  modrow(1,0);            <<< this defines module 1 row 0 on this connector (bits
 are defined in the device definition to define the physical
 location of the switch.)
          modrow(1,1);
  modrow(1,2);
  modrow(1,3);
  modrow(1,4);
.....other module row definitions 5 though 15
};
```

connector(Econ) // this will be a VIRTUAL connector. Virtual connectors are used for event passing and are not real module ,rows. Virtual connectors start at module 16

```
{
analog(0);.....thru analog(15);  to allow analogs to be defined for devices using event
connector (Econ)
modrow(16,0);
};
```

//event device first.

This will correspond to the first device (first EPIC USB in Game Controllers) and corresponding [JOYSTICK_MAIN .....] section in FS2002.CFG.

This section "usually" has an 8001 in the GUID if EPIC was installed before other joysticks.

```
    device(Events0)
    {
      connector(Econ);
      analog(0,Ailerons,X);
      button(0,1,GEAR_UP);
```

this corresponds to the FS2002.CFG definition and recommend using the same FS2002 keyword. See Dowson FS2002 docs for list of keywords.

```
    BUTTON_DOWN_EVENT_01=GEAR_UP
    button(0,2,GEAR_DOWN);  <<< BUTTON_DOWN_EVENT_02=GEAR_DOWN
.....other event definitions
    };
.....other event passing devices.
.....devices here to pass events to DLL if desired

//physical devices
//
    device(Firewall)
    {
      connector(Mod1);
      button(4,3,GearSw);     <<<<module1 row4 bit 3
      .....other button definitions
    };
```

Now that the device and button name are defined, these can be used by name using the format DeviceName.ButtonName or DeviceName.AnalogName in the EPL source code.

The switch can be linked to a procedure by using that name and .On or .Off

```
:Firewall.GearSw.On{Events0.GEAR_UP = pulse;}   // when the gear switch goes on
    (UP) send the Events0.GEAR_UP event.
```

This will pulse button 1 on device 0 and FS2002 will see this

LESSON 1:

Notes:

This will only work correctly in Windows 2000 and Windows XP

Windows 98 first edition does not send the data to the correct place.

Windows98SE will work with fixed descriptors, but the devices may come out in the wrong order, or it may put the analogs on the wrong devices.

WindowsME mixes up the devices with each unplug/replug.

We may go back and attempt to work around these problems in the future with a registry edit, unless Microsoft fixes these problems.

Game Controllers buttons are labeled starting at 1. References to button numbers actually start at 0. So device button0 shows up as 1.

Windows allows a maximum of 8 axes, 32 buttons, and 1 POV per device.

This may change in the future.

This document assumes you have a cheap joystick plugged into the First Expansion module "A" connector and that the analog axes have been "modified" to be 3 wire. Two wire analogs are no longer supported.

Set unplug options:

Options | loader

There should be a check mark on "Unplug on load"

Unplug Delay set to 500

Replug Delay set to 1000

Operating System Delay 4000

You can experiment with these values for faster loading. The most important is the "Operating System Delay"

When EPICUSB unplugs itself and replugs the system has to load driver, re-enumerate, and build registry entries.

If this delay is too short, EPICenter may attempt to read info from EPICUSB before everything is ready and fail.

Some systems can use 2000 (2 seconds), usually 3000 is safe, 4000 should be really safe.

Create a project. :

Project | new

project name TestDesc

Accept that the .EPL file was not found.

In the project window, click testdesc.joy, right click, regenerate

TestDesc.EPL was minimized and probably up in the left corner, click on the expand box,

between the "-" and the "x" (you may have to click on the "X" to close, then reopen from project window by double clicking on TestDesc.EPL).

Type:

```
#include <EPICDEFS.HPL>          //contains common definitions and in EPIC\include
                                 // directory
definemodule(0,FASTSCAN,0,7)     //scan module 0 (first expansion module, A,B,
                                 // RUDDER, and  FLCS connectors
#include "mydevices.hpl"          //the "  " says to look in the current directory.
// < mydevices.hpl>
// would look in \EPIC\include directory


:INIT
{
}

on the menu bar:
File | new
```

type the following into the window:

```
#pragma hid_rpt_snd    1          //this is the number of devices to send to Direct
Input                                                    //these devices will
appear in Game Controllers.
connector(FirstExpansion)
{
analog(0);                       //input analogs
analog(1);
analog(2);
analog(3);

modrow(0,0);          // "A" connector on First Expansion module
};

connector(Econ0)      //events connector. This is a virtual connector and is not real. This is
                      // used in devices
```

```
// to send events to Direct Input
    {
      analog(0);
      analog(1);
      analog(2);
      analog(3);              //all analogs can be placed here if needed

      modrow(16,0);           //There is no module 16, this is a virtual module
    };                        //<<<<<note the terminating ";"

//*****Direct input devices **********

    device(Events0)
    {
      connector(FirstExpansion);
      analog(0,roll,X);   // source analog0, X axis on first EPIC USB device in Game Controllers

      button(0,0,B0);    //button0 (this will show up in Game Controllers marked as "1"
    };
```

click File | save as mydevices.hpl

Click the "cascade" button on the toolbar to see all windows.
click project | make or hit F9 key
you should get "done" on the last line. If not, find the typing error, correct and repeat.
click project | load or hit F10 key
you should get "Loaded and verified"

Minimize "EPICenter" by clicking on the "-" on the EPICenter main window bar upper right corner double click on "Game Controllers" If you do not have a short cut to Game Controllers create one by going to Start | Settings | Control Panel | Game Controllers and drag to desktop.

You will see one EPIC USB device in Gaming controllers.
Double Click the EPIC USB device.
You will see 1 X axis in the box and one button.

IF you have a joystick (with a 3 wire X axis (2 wire not supported, see Notes on 2 wire "STANDARD")) and a trigger, you will see the X axis move and the 1 button come "on" when the trigger is depressed and go off when the trigger is released. If you do not see the axis move it may need to be calibrated. Calibrate in Game Controllers with the "view raw data" checked.

IF a procedure was assigned to this button,for example:

:Events0.B0.On{ body of procedure to execute when Switches0.B0 went "on" }

you would NOT see the button action.

LESSON 2

Change device definition by adding 2 analogs and 4 more buttons:

Make the following changes to mydevices.hpl :

```
//****** Direct input devices ******
      device(Events0)
      {
        connector(Econ0);   // change the connector to the virtual connector "Econ0"
        analog(0,roll,X);     // source analog0, X axis on first EPIC USB device in Game Controllers
      analog(1,pitch,Y);
      analog(3,Throttle,Z);

      button(0,0,B0);              //button0 (this will show up in Game Controllers marked as "1"
      button(0,1,SecondButton);
      button(0,2,ThirdButton);
      button(0,3,B3);
      button(0,4,B4);
      };
//**************physical devices *************
   //This will not show up in Game Controllers since hid_rpt_snd = 1 and this is the second device

      device(Switches0)
      {
      connector(FirstExpansion);
         button(0,0,Trigger);
      };
```

Save

Project | Make

Project | Load

minimize EPICenter

Wait for TP2 led (second from left) on EPICUSB to go back out

Double click Game Controllers

You will see one EPIC USB device even though you have two devices defined since hid_rpt_snd=1

double click EPIC USB. You will see the XY box and a Z axis, and 5 buttons.

LESSON 3

Add a POV device:

```
(Events0)
{
  connector(Econ0);      // change the connector to the virtual connector "Econ0"
  analog(0,roll,X);      // source analog0, X axis on first EPIC USB device in
                         // Game Controllers
  analog(1,pitch,Y);
  analog(3,Throttle,Z);

  button(0,0,B0);   //button0 (this will show up in Game Controllers marked as "1"
  button(0,1,SecondButton);
  button(0,2,ThirdButton);
  button(0,3,B3);
  button(0,4,B4);

  pov(view);                    //"view" is the name of the POV element for this device.
};
```

This is a point of view with a range of 0 to 359.00 degrees.

Save, make, load, Game Controllers

double click EPIC USB

You now have 3 axes, 5 buttons, and 1 point of view.

Controlling the POV:

In EPICenter:

Window / TestDesc.epl

IF TestDesc.epl is not in the list, double click TestDesc.epl in the project window, or file / open and select from dialog box.

add

```
: Switches0.Trigger.On{Events0.view = 9000;}  //when trigger button goes on move to 90°.
                                              //     position is in 1/100 of a degree
: Switches0.Trigger.Off{Events0.view = 0;}        // when trigger button goes off move
                                                  // pov to 0 degrees
```

save, make,.....

in Game Controllers EPICUSB you will now see 3 axes, 5 button, 1 POV

When the trigger is depressed the POV will move to 90 degrees, when released the POV will move back to 0 degree position.

To see the button change in Game Controllers device:

Edit TestDesc.epl

Change:

```
: Switches0.Trigger.On
 {
Switches0.B0 = on;              // turn on Device 1 button0
                                //      this is the same as nqw(BtnOn,0x0100);
Switches0.view = 9000;          //position is in 1/100 of a degree
}


: Switches0.B0.Off
{
Switches0.B0 = off;             //turn off Device 0 button 0
                                //this is the same as  nqw(BtnOff,0x0100);
Switches0.view = 0;             // position to 0 degrees
}
```

save,make,.....

Now the pov will move to 90 AND Button0 (marked "1") of the device will come on when the trigger is depressed.

The pov will move to 0 AND Button0 will go off when the trigger is released.

LESSON 4

Add another device.

Add a virtual analog.

Under the descriptor for Switches0

```
device(Events1)
{
connector(Econ0);
analog(2,Rudder,Rx);
//16 bit analog named "virtual" on axis Rz with range 0-0xFFFF
analog(VIRTUAL, Virtual, Rz, MEM16, , 0x0000, 0xFFFF);
button(0,5,TB0);
button(0,6,B1);
button(0,7,B2);
};
```

Edit TestDesc.epl:

change procedure to be executed for trigger:

```
:Switches0.Trigger.On{jump(CalibrateVirtual);}
comment out the off action by:
/*                  <<<<<start of block comment
: Switches0.B0.Off
{
Switches0.B0 = off;                 //turn off Device 0 button 0
                                    //this is the same as  nqw(BtnOff,0x0100);
Switches0.view = 0;                 // position to 0 degrees
}
*/                  <<<<<<<end of block comment

:CalibrateVirtual
{
SecondDevice.virtual = 0;
delay(10);
SecondDevice.virtual = 0x8000;          //half way
```

```
delay(10);
SecondDevice.virtual = 0xFFFF;          //all the way
delay(10);
if(Switches0.B0) jump CalibrateVirtual;
SecondDevice.virtual = 0x8000;          //stop at half way
}
```

make (F9)

You will get an error "unknown label (CalibrateVirtual)

This is because the procedure "CalibrateVirtual" was defined AFTER the procedure that attempted to execute it and the compiler did not know "CalibrateVirual" was below it. To solve this problem, you can move the CalibrateVirtual procedure BEFORE the Switches0. Trigger.On procedure, or make a forward declaration in a header file (.hpl). The header files include definitions of procedures in the source code and should have the same name as the source file (.epl file);

File | new

in the blank window

```
:CalibrateVirtual;<<<note the semicolon at the end
```

This can also be in the "C" format <return Type> <ProcedureName> (<argumentList>). At this time we only support the "void" type.

void CalibrateVirtual(void); means no return value and no argument list and is the same as :CalibrateVirtual;

File | save as TestDesc.hpl

In TestDesc.epl

add

```
#include "TestDesc.hpl"
```

under the #include "mydevices.hpl"

for multiple files you should group the #include .hpl files above the #include .epl file list.

F9 to compile F10 to load

minimize EPICenter and double click Game Controllers.

You will see 2 EPIC USB devices now.

Double click the second one and you will see two axes, Rx and Rz, and 3 buttons.

Calibrate with raw data showing.

When calibrating the Virtual Rz axis, activate trigger and you will see the axis going from 0 to 0x8000 to 0xFFFF (0 to 32736 to 65535)

This is the end for now, more will be added in the future.

Then this could be changed to:

```
defbtn(GEAR , on,GearUp)
defbtn(GEAR, off, GearDown)

:GearUp {nqw(BtnP,GEAR_UP)}
:GearDown       {nqw(BtnP,GEAR_DOWN)
```

Which is easier to read.