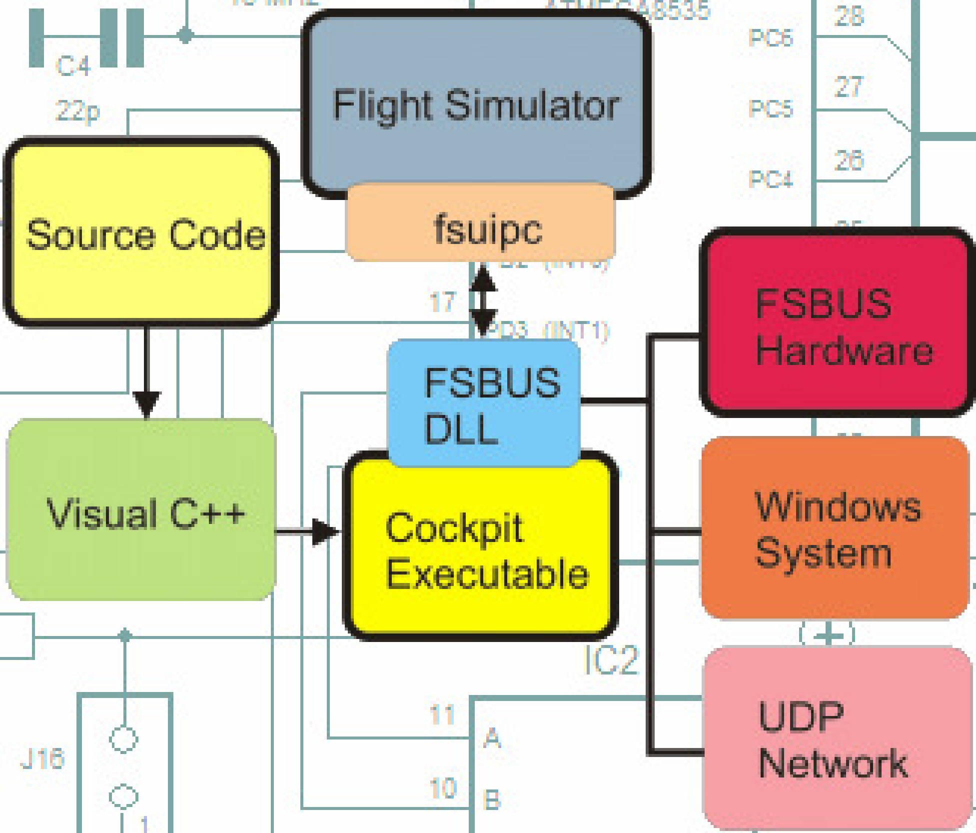
FSBUS

DLL Reference Manual



Terms of use

The software and documentation included in the FSBUS DLL is copyrighted by Dirk Anderseck (dirk@anderseck-net).

Permission is granted to anyone to use this software for any purpose, FSBUS DLL is free software for non-commercial purpose.

All rights in the Software (including any images or text incorporated into the Software) are owned by the author of this software, except otherwise stated.

The author of this software does not guarantee that this software is free from bugs or free from viruses.

The author may, from time to time, revise or update the Software. In so doing, the author incurs no obligation to furnish such revision or updates to you.

The software is provided “as is” and the author disclaims all other warrenties and conditions, fitness for a particular purpose, conformance with description, title and non-infringement of third party rights.

Upgrade information

Changes and improvements in Version 2

In Version 2 you may define more than one callback function.

Changes and improvements in Version 3

Code changes

11/2010 fbus.dll Version 3 is now in pure C-style to support the Borland compilers.
The following functions are affected

Version 2 format	Version 3 format
`BOOL FsWrite(int oid, int i32)`	`BOOL FsWriteInt (int oid, int i32)`
`BOOL FsWrite(int oid, double d`	`BOOL FsWriteDbl (int oid, double d)`
`BOOL FsWrite(int oid,__int64 i64)`	`BOOL FsWriteInt64 (int oid, __int64 i64)`
`BOOL FsWrite (int oid, UVAR u)`	`BOOL FsWriteUnion (int oid, UVAR u)`
`BOOL FsbusWrite(unsigned char* buf, int count)`	`BOOL FsbusWriteRaw(unsigned char* buf, int count)`
`BOOL MkFsObject (int oid,char* nm, void(* cb)(int oid, int val, double dbl), DWORD offset,int datasz, UTYPE datatp, FSINTERVAL intvl, int flags = 0);`	`BOOL MkFsObject (int oid, char* nm, void(* cb)(int oid, int val, double dbl), DWORD offset,int datasz, UTYPE datatp, FSINTERVAL intvl, int flags);`
`BOOL MkFsbusObject(FSBUSTYPE ftp, int oid, char* nm, void(* cb)(int oid, int val, double dbl), int c, int r, int flg = 0);`	`BOOL MkFsbusObject(FSBUSTYPE ftp, int oid, char* nm, void(* cb)(int oid, int val, double dbl), int c, int r, int flg);`

Exceptions

The error handling mode EM_THROWEXCEPTION is removed. This doesn't work in a pure C environment.

FsInvalidate()

A new function FsInvalidate() is added to the library. This is a convenient way to get request all events from flightsim.

Battery and Power

Handling low power conditions is now implemented inside DLL (optional). This makes application development a lot easier.
Example:

You create an Digital Out object to drive an LED (OID_LED).
In case of a power off condition on the mainbus, you want to turn off the LED.

```
The code to achieve this is:
      FsbusPowerOffOptions (OID_LED, PWRTYPE_MAINBUS, 0);
```

A zero will be sent to the LED, when a power off condition occurs.

The powerstate itself must be notified to the DLL by:

```
SetPower(PWRTYPE_MAINBUS,0);
   -  or -
SetPower(PWRTYPE_MAINBUS,1);
```

Normally this takes place in a callback function, where the MAINBUSVOLTAGE is notified.
PWRTYPE_MAINBUS is only one of many possible power bus systems.
PWRTYPE_AVIONIC is a second predefined bus and many others can be individually created.

Sound

If you like to play a wav file for a long time, you may play it in a loop.

```
MkLoopSound (22, „myMotor“, NULL, „motor.wav“);
```

After the object begins playing it will loop endless until you make a call to **Stop**.

Enable / Disable

The Enable() and Disable functions are improved. So far the concerned object was excluded or included from the polling list to flightsim. From this version on, all traffic, in- and outgoing is controlled by these functions.

FSBUS DLL with Borland C++ Builder

The Borland compilers offer many advantages for Rapid Application Development. The FSBUS DLL version 3 is now ready to be used with these compilers and linkers. The following steps are required to implement it into a project.

- Copy the fsbus.h and fsbus.dll into your project directory
- Call **implib -a fsbus.lib fsbus.dll** on the command line
- Add the include statement in any of the source files: **#include "fsbus.h"**
- Add **#pragma comment(lib, "fsbus.lib")** into the main source file

Introduction

This SDK is posted with the intention that flightsim enthusiasts around the world will be able to build their own cockpit that are both inexpensive and highly functional. While the instructions may be printed and followed at no charge, they are not intended to be used as instructions for starting your own business.

The FSBUS DLL is a powerfull tool for your home-cockpit. It integrates the flightsim interface, the FSBUS hardware, the DirectX Sound Mixer, timers and more into a programming API.

You need to define your FSBUS hardware objects and modify the list of the fsuipc variables, which are predefined for your convinience.

There is plenty of space to add variables of 3rd party software.

I tried to make the programming task as easy as possible. That’s the reason why the syntax is classic C-style for console applications.

I have developed the DLL with Microsoft Visual C++ 2008 Express, which is free available in the internet. The tests were made with Windows Vista and FSX SP2. It may also work with Windows XP and FS2004.

The current FSBUS DLL makes use of fsuipc. According to the license policy of the fsuipc interface, your selfmade application may require a personal license, which you can refer over the Internet. It’s up to you to fullfill these license terms.

DLL Overview

BCD2Int	Converts a bcd value to an integer
Calibrate	Converts an integer by a table and interpolate
CheckIn	Initialize the DLL
CheckOut	Free the DLL
Disable	Disable an object
DisplayOptions	Setup a 7 segment display
Enable	Enable an object
Error	
ExtKeyboard	Send control key sequences and mouse events
FsbusClose	Close the FSBUS interface
FsbusMux	Handle all events
FsbusOpen	Open the FSBUS interface
FsbusPowerOffOptions	Defines beheavior on power off
FsbusWrite	Write a buffer to FSBUS electronic
FsbusWriteFmt2	Write a value to FSBUS electronic in 2 byte format
FsbusWriteFmt3	Write a value to FSBUS electronic in 3 byte format
FsbusWriteFmtVar	Write a value to FSBUS electronic in variable length format
FsConnect	Connect to the fsuipc interface
FsDisconnect	Disconnect from fsuipc interface
FsReadDirect	Read a value from fsuipc
FsSetPollTiming	Modify the poll timing
FsWriteInt	Write to flightsim
FsWriteDbl	Write to flightsim
FsWriteUnion	Write to flightsim
FsWriteInt64	Write to flightsim
FsWriteDirect	Write immediatly to flightsim
FsInvalidate	Invalidates all flightsim objects. As a result, the application will be notified the next time, this parameter is polled.
GetFsbusDLLVersion	Get the Fsbus version information
Int2BCD	Convert an integer to a bcd value
Keyboard	Send a simple key command
MkFsObject	Create a flightsim object
MkFsbusObject	Create an fsbus object
MkUdpInterface	Create a udp channel
MkTimer	Create a timer object
MkSound	Create a multimedia polyphonic sound object
MkLoopSound	Create a multimedia polyphonic sound object playing in a loop.
Pan	Position the sound between left and right
Play	Start playing a sound object
Rewind	Rewind the sound object
SetPower	Notifies the DLL about certain power conditions
SetTime	Modify a timer
Stop	Stop the sound object.
UdpDestination	Define the Destination for udp channel

UdpSend	↗	Send a buffer via udp
Vol	🔈	Set the volume of a sound object

● Core
↗ Network
ABC new

✈ Flightsim
🔈 Sound

FSBUS Fsbus
💻 System

Alphabetic function listing

BCD2Int

Syntax

```
int BCD2Int (unsigned short bcd)
```

This helper function converts the bcd based dataformat of some fsuipc variables. Some values of navigation frequency values have a bcd format.

Return

Example

BCD Value	Return value
0x35	0x23 → decimal 35
0x10	0x0a → decimal 10
0x512	0x200 → decimal 512

Calibrate

Syntax

```
int Calibrate (int val, CALTAB* t, int count)
```

Sometimes it is necessary to convert a value from fsuipc into a special output value. Think about servo applications, where the scale of a gauge is nonlinear or the servo itself requires some corrections.

This function interpolates a value out of a table which you can supply. You can add an arbitrary number of points to the table.

The input value must be in ascending order!

Example

```
Example 1:

the input values from 10-250 are interpolated to 0-16000

 CALTAB t[] = { {10, 16000},{ 250, 0} };
 int x = Calibrate (50, t, sizeof(t)/sizeof(LITAB));


Example 2:

An fsbus control (C_XY) sends a value from 10 to 220. You translate it
into an output value from 0 to 16000.

void EventCallback (int oid, int v)
{
    static CALTAB t[] = { {10, 0}, {40, 3000}, {70, 5800},
                  {100, 8500}, {130, 11000}, {160, 14000},
                  {190, 15000}, { 220, 16000} };

    switch (oid)
    {
    case C_XY:
        Fsrite (FS_CTRL, Calibrate(v, t, 8);
        break;
    }
}
```

CheckIn

Syntax

```
BOOL CheckIn ()
```

This initializes the DLL functions and objects. cbEvent Is a pointer to a function which receives any event of flightsim, fsbus hardware and timers.

Return

This function will return, if it was success full. If an error occurs the error function is executed.

Unless you change the default error mode, the application will show a message and stop execution.

Example

```
int main(int argc, char* argv[])
{
    CheckIn();
}
```

CheckOut

Syntax

BOOL CheckOut ();

This should be called before your application exits.

Example

```
int main(int argc, char* argv[])
{
    …
    CheckOut();
    return 0;
}
```

Disable

Syntax

```
BOOL Disable (int oid)
```

This function disables an object.

Example

DisplayOptions

Syntax

```
BOOL DisplayOptions (int oid, int SegCount, int SegOffs, bool LeadZero, int DecPoint);
```

Display Options sets some paramters to a 7 segment display controller. The number of segment can vary between 3 and 6, the offset parameter is a shift of the display start. If you want leading zeros, set LeadZero to TRUE, The DecPoint controls the position of decimal point, where the value zero means no decimal point.

Example

```
#define OID_D1  1

MkFsbusObject (
               BTP_DISPLAY,
               OID_D1,         // unique object id
               NULL,           // the optional user readable name
               NULL,           // the callback function
               1,              // Controller ID of FSBUS card
               0);             // Register ID not used for displays

DisplayOptions (
               OID_D1,         // unique object id
               5,              // number of segments in display
               0,              // an optional offset of the segments
               true,           // leading zeroes
               0);             // the decimal point position is off

// loop to write 1000 numbers with a delay of 10ms between each
for (int i=0; i<1000; i++)
{
    FsbusWrite (OID_D1, i);
    FsbusMux(10);
}
```

Fsbus DLL is a highly optimized engine to route the data between the endpoints. Each of the FsbusWrite functions puts the data into a queue in memory. Without calling FsbusMux(), it will never be sent to the bussystem. This is automatically done in normal operation, where FsbusMux is the heart of your code, but if you are using the debugger of Visual Studio, you may not see an effect on the display before FsbusMux is called.

Enable

Syntax

`BOOL Enable (int oid)`

This function enables an object.

Example

Error

Syntax

`void Error(LPSTR fmt, ...)`

You can use this function, to display an error on console. After display, the normal fsbus error handling procedures are processed.

The error mode is controlled by the variable **ErrorMode**.

The modes are EM_STOPEXECUTION (default) or EM_RESUMERETURN.

The reason for an fsbus based error can be retrieved by reading the variable **ErrorText**.

Example

ExtKeyboard

Syntax void ExtKeyboard (const char* complex);

If you need to generate other keys, modifier keys, mouse events or other controls, you'll have to use the complex format. A command in complex format consists of a letter, followed by optional + or – and parameters. Strings containig more than one command have a semicolon acting as command separator.

The control characters:

K+	A key down event is pushed on the windows keyboard queue
K-	A key up event is pushed on the windows keyboard queue
L+	A left mouse down event is pushed on the windows mouse queue
L-	A left mouse up event is pushed on the windows mouse queue
R+	A right mouse down event is pushed on the windows mouse queue
R-	A right mouse up event is pushed on the windows mouse queue
D	Delay between 2 events

Return
Example

K+c	Key Down Key = 'c'
K-c	Key Up key = 'c'
L+30,40	left mouse down X=30 Y=40
L-30,40	Left mouse up X=30 Y=40
R+100,400	Right mouse down X=100 Y=400
R-10,30	Right mouse up X=10 Y=30
D50	Delay 50ms
K+VK_SHIFT;K+c	Press Shift key, press 'c'
K-c; K-VK_SHIFT	Release 'c', release Shift

List of windows keynames

VK_CANCEL	VK_BACK	VK_TAB	VK_CLEAR
VK_RETURN	VK_SHIFT	VK_CONTROL	VK_MENU
VK_PAUSE	VK_ESCAPE	VK_SPACE	VK_PRIOR
VK_NEXT	VK_END	VK_HOME	VK_LEFT
VK_UP	VK_RIGHT	VK_DOWN	VK_LCONTROL
VK_RCONTROL	VK_LMENU	VK_RMENU	VK_SELECT
VK_PRINT	VK_EXECUTE	VK_INSERT	VK_DELETE
VK_HELP	VK_LWIN	VK_RWIN	VK_APPS
VK_SNAPSHOT	VK_SLEEP		
VK_NUMPAD0 -	VK_NUMPAD9		
VK_MULTIPLY	VK_ADD	VK_SEPARATOR	VK_SUBTRACT
VK_DECIMAL	VK_DIVIDE	VK_NUMLOCK	VK_SCROLL
VK_F1 -	VK_F24		
VK_OEM_NEC_EQUAL	VK_LSHIFT	VK_RSHIFT	VK_BROWSER_BACK
VK_BROWSER_FORWARD	VK_BROWSER_REFRESH	VK_BROWSER_STOP	VK_BROWSER_SEARCH
VK_BROWSER_FAVORITES	VK_BROWSER_HOME	VK_VOLUME_MUTE	VK_VOLUME_DOWN
VK_VOLUME_UP	VK_MEDIA_NEXT_TRACK	VK_MEDIA_PREV_TRACK	VK_MEDIA_STOP
VK_MEDIA_PLAY_PAUSE	VK_LAUNCH_MAIL	VK_LAUNCH_MEDIA_SELECT	VK_LAUNCH_APP1
VK_LAUNCH_APP2	VK_OEM_1	VK_OEM_PLUS	VK_OEM_COMMA
VK_OEM_MINUS	VK_OEM_PERIOD	VK_OEM_2	VK_OEM_3
VK_OEM_4	VK_OEM_5	VK_OEM_6	VK_OEM_7
VK_OEM_8	VK_OEM_AX	VK_OEM_102	VK_ICO_HELP
VK_ICO_00	VK_PROCESSKEY	VK_ICO_CLEAR	VK_PACKET
VK_OEM_RESET	VK_OEM_JUMP	VK_OEM_PA1	VK_OEM_PA2
VK_OEM_PA3	VK_OEM_WSCTRL	VK_OEM_CUSEL	VK_OEM_ATTN
VK_OEM_FINISH	VK_OEM_COPY	VK_OEM_AUTO	VK_OEM_ENLW
VK_OEM_BACKTAB	VK_ATTN	VK_CRSEL	VK_EXSEL
VK_EREOF	VK_PLAY	VK_ZOOM	VK_NONAME
VK_PA1	VK_OEM_CLEAR	VK_CAPITAL	VK_KANA
VK_HANGUL	VK_JUNJA	VK_FINAL	VK_HANJA
VK_KANJI	VK_CONVERT	VK_NONCONVERT	VK_ACCEPT
VK_MODECHANGE			

There is a helper tool integrated in the fsadmin program, which helps finding the correct sequences.

FsbusClose

Syntax

BOOL FsbusClose();

Close the serial interface.

Example

```
FsbusClose();
```

FsbusMux

Syntax BOOL FsbusMux (int maxtime);

Return

This is the single multiplexer function, which handles all your defined objects. Usually you call this function in a permanent loop. In order to get you a chance to add own code, you define a time in milliseconds. The function will return after that period.
In the example below, the kbhit function checks for a keyboard action after the 500ms FSBUS actions are done.

Don’t worry about 500ms actions with full cpu load, the strategy is to sleep most time. It’s your responsibility to keep your event code as efficient as possible.

```
while (!kbhit())
    FsbusMux(500);
```

FsbusOpen

Syntax BOOL FsbusOpen (LPSTR dev);

Open the serial interface. Dev is one of the predefined device names like “COM1” “COM2” a.s.o.

If FsbusOpen fails, an exception is thrown with a text message, which probably shows a reason.

Example

```
    Try
    {
        CheckIn(EventCallback, EventCompare);
        FsbusOpen("COM1");
    }
    catch (LPSTR text)
    {
        cout << text << endl;
    }
```

FsbusPowerOffOptions

Syntax void FsbusPowerOffOptions (int oid, int powerbustype, int offvalue);

This function defines an optional power-off value for an fsbus object, which is used in case of a specific powerstate.

Example

This example describes a full power scenario for a display.

```
// 1. Create a display object
MkFsbusObject (
      BTP_DISPLAY,
      OID_D1,            // unique object id
      NULL,              // the optional user readable name
      NULL,              // the callback function
      3,                 // Controller ID of FSBUS card
      0,                 // RID
      0);                // flags

// 2. Set the optional value for a power-off situation
FsbusPowerOffOptions (OID_D1, PWRTYPE_MAINBUS, DISPLAY_BLANK);

// 3. Create a fs object to get power notifications
MkFsObject (FS_MAINBUSVOLT,"",cb,0x2840,8,TP_DBL,FS_NORMAL,0);

// 4. Write the callback function
void cb(int oid, int val, double dval)
{
  static bool mainbuson = false;

  switch (oid)
  {
  case FS_MAINBUSVOLT:
      printf ("Uges: %lfV\r\n", dval);
      if (mainbuson && (dval < 21.0))
      {
            mainbuson = false;
            SetPower(PWRTYPE_MAINBUS, 0);
      }
      if (!mainbuson && (dval > 21.5))
      {
            mainbuson = true;
            SetPower(PWRTYPE_MAINBUS, 1);
      }
      break;
  }
}
```

The SetPower() function will lookup all fsbus objects, which have a power-off value set (FsbusPowerOffOptions) and send the value automatically to the hardware controller.
This makes application development much more convinient.

Each control may have a power-off value defined for a specific power source. PWRTYPE_MAINBUS and PWRTYPE_AVIONIC are predefined. You may extend the types by simply using a value > 2.

FsbusWrite

Syntax

BOOL FsbusWrite (int oid, int val);

FsbusWrite is used to send a value to an fbus object.

This function will store the data onto a queue. Only a call to FsbusMux will send the data onto the bussystem. This is important to know, when you are working with the debugger in single step mode.

Example

```
FsbusWrite (C_VOR1, 110500);
```

FsbusWriteRaw

Syntax BOOL FsbusWriteRaw(unsigned char* buf, int count);

Usually, you send a command to one of your cockpit controls by FsbusWrite(int oid, int val).

FsbusWrite is a generic function to build and send a command buffer by yourself. Please refer to the software interface description at the end of this book.

This function will store the data onto a queue. Only a call to FsbusMux will send the data onto the bussystem. This is important to know, when you are working with the debugger in single step mode.

Example Send a reset command as broadcast to each controller.

```
char buf[2];
    // declare a 2 byte buffer
buf[0] = 0x81;
    // Bit 2^7 is always 1. 2^0 is the first value bit
buf[1] = (128 & 0x7f);
    // 128 is the R-Command Reset
FsbusWriteRaw (buf, 2);
```

You can also send this command with FsbusWriteFmt2()

FsbusWriteFmt2

Syntax BOOL FsbusWriteFmt2(int cid, int rid, int val);

This is another way to send a command to an fsbus controller. All 2 byte long commands are sent with this function.

This function will store the data onto a queue. Only a call to FsbusMux will send the data onto the bussystem. This is important to know, when you are working with the debugger in single step mode.

Example Send a reset command as broadcast to each controller.

```
FsbusWriteFmt2 (0, 128, 1);
```

FsbusWriteFmt3

Syntax BOOL FsbusWriteFmt3(int cid, int rid, int val);

This is another way to send a command to an fsbus controller. All 3 byte long commands are sent with this function.

This function will store the data onto a queue. Only a call to FsbusMux will send the data onto the bussystem. This is important to know, when you are working with the debugger in single step mode.

Example Send the voltage information (80%) to all controllers with command 131.

```
FsWriteFmt3 (0, 131, 80);
```

FsbusWriteFmtVar

Syntax BOOL FsbusWriteFmtVar(int cid, int rid, int val);

This Write function sends a variable length data format to controllers which require longer data formats like stepper controller.

This function will store the data onto a queue. Only a call to FsbusMux will send the data onto the bussystem. This is important to know, when you are working with the debugger in single step mode.

Example

FsConnect

Syntax BOOL FsConnect ();

Connects the object to fsuipc. Flightsim must be started before executing FsConnect.

Return

Example

FsDisconnect

Syntax BOOL FsDisconnect ();

Disconnects the object from fsuipc.

Return

Example

FsInvalidate

(3.0.1)

Syntax void FsInvalidate ();

All flightsim objects are requested to provide the application with it's value, when the next poll occurs.

Return

Example

FsReadDirect

Syntax `BOOL FsReadDirect (int offset, int sz, void* dest)`

You can read any offset in fsuipc without declaring an object with the FsReadDirect function.

If you are reading hundreds of variables, FsReadDirect is less efficient.
Use the polling method whenever possible. ReadDirect is useful if you read a value only once (Versionnumber, …) or if you need to read an exceptional dataformat.

Return

Example

```
short zoom;
FsReadDirect (0x02b2, 2, &zoom);
```

FsSetPollTiming

Syntax `BOOL FsSetPollTiming (int quick, int normal, int lazy)`

Any created DFsObject is assigned a poll class . The class determines the pollfrequency.
There are 4 classes defined:
None (no polling)
Lazy (3s)
Normal (300ms)
Fast (100ms)

You can modifiy these times with `SetPollTiming`.

Example

```
FsSetPollTiming (50, 500, 5000);
```

FsWriteInt

(3.0.0)

Syntax `BOOL FsWriteInt (int oid, int )`

This is a write function to send an integer value to a flightsim object.
The update occurs during the next poll sequence.

Return

Example

FsWriteDbl

(3.0.0)

Syntax `BOOL FsWrite (int oid, double)`

This is a write function to send a double value to a flightsim object.
The update occurs during the next poll sequence.

Return

Example

FsWriteInt64

(3.0.0)

Syntax
```
BOOL FsWrite (int oid, __int64)
```

This is a write function to send a 64 bit integer value to a flightsim object.
The update occurs during the next poll sequence.

Return

Example

FsWriteUnion

(3.0.0)

Syntax
```
BOOL FsWrite (int oid, UVAR )
```

This is a write function to send a union value to a flightsim object.
The update occurs during the next poll sequence.

Return

Example

FsWriteDirect

Syntax
```
BOOL FsWriteDirect (int offset, int sz, void* dest)
```

Updating a variable in flightsim is usually done in the poll procedure.
If you like to write something to flightsim, which you haven't declared as an object, FsWriteDirect is your choice.

Return

Example
```
short zoom = 100;
FsWriteDirect (0x02b2, 2, &zoom);
```

GetFsbusDLLVersion

Syntax
```
int GetFsbusDLLVersion ()
```

Return Return value is a 3 digit version number of this DLL.

Example
```
int v =  GetFsbusDLLVersion ();

printf ("Fsbus DLL version = %d.%d.%d", v/100, (v/10)%10, v%10);
```

Int2BCD

Syntax

unsigned int Int2BCD (int i);

```
This function converts an integer to a bcd based dataformat of some
fsuipc varaibles mainly in the navigation area.
```

Return

Example

Keyboard

Syntax

void Keyboard (const char* simple);

The FSBUS simple keyformat can generate keycodes for the following keys:

Numeric	0-9
Lowercase alpha	a-z
Upercase alpha	A-Z
German Umlauts	ä ö ü ß
Special characters	# , . - ´ < +

Return

Example

```
Keyboard("A");
```

MkFsObject

Syntax

```
BOOL MkFsObject (
            int oid,
            char* name,
            void(* cb)(int oid, int val, double dbl),
            DWORD offset,
            int datasz,
            UTYPE datatp,
            FSINTERVAL intvl,
            int flags);
```

```
This function creates an object for the flightsim interface. You do
this by specifing and id, offset and length (fsuipc programmers
manual), a datatype, one of 4 speed classes and optional flags.
```

Parameter

```
Oid: a unique id for this object. This will be a systemwide access key
to the new created object.

Name: an optional name for this object, which is shown in case of an
error message.

CB: the callback function which receives event messages from this
object. If the object will not send any event of interest, you can
specify NULL.

Offset: this defines the offset in the fsuipc interface.
Datasz: the length of the parameter in byte according to fsuipc
interface. The length is restricted to a maximum of 8 byte. There are
longer variables in fsuipc. These need to be read by FsReadDirect.

Datatp: the type of the data. It is used for a default conversion into
an int32 datatype. Valid types are:
    TP_I8, TP_UI8,       // signed, unsigned 8Bit vars
    TP_I16, TP_UI16,     // signed, unsigned 16Bit vars
    TP_I32, TP_UI32,     // signed, unsigned 32Bit vars
    TP_I64,              // signed 64Bit vars
    TP_DBL               // 8Byte floating point vars

Intvl: there are 4 classes to define the poll frequency
FS_NONE,            // no polling (reading)
FS_LAZY,            // one read in 3s
FS_NORMAL,          // one read in 300ms
FS_QUICK            // one read in 100ms
( you can change the timing by callung FsSetPollTiming)

The only flag for FS objects is the FLG_DISABLED flag, which turns the
object off until you reenable it.
```

Return

```
TRUE, if the object was created successfull.
```

Example

```
#define F_BUSVOLT  1002

MkFsObject (F_BUSVOLT, cb, “BUSVOLT”, 0x2384,8, TP_DBL, FS_LAZY, 0);
```

MkFsbusObject

Syntax

```
BOOL  MkFsbusObject (
                  FSBUSTYPE ftp,
                  int oid,
                  char* name,
                  void(* cb)(int oid, int val, double dbl),
                  int c,
                  int r,
                  int flg=0);
```

BTP_D_IN	Digital Input on the IO controller
BTP_ROTARY	Rotary Input on the IO controller
BTP_A_IN	Analogue input on the IO controller
BTP_D_OUT	Digital Output on the IO or DO64 controller
BTP_DISPLAY	7 segment display controller
BTP_A_OUT	Analogue Output on IO or servo controller the first channel begins with r = 80, the 8th channel is r = 87 This object supports a value range from 0 to 255.
BTP_V_OUT	Stepper controller the first channel begins with r = 80, the 8th channel is r = 87 This object supports a value range from 0 to 4Giga.

Example

```
#define C_GEARUP   3001

int main(int argc, char* argv[])
{
    MkFsbusObject(BTP_D_IN,C_GEARUP,"GEARUP",EventCallback, 2, 33);
}


void EventCallback (int oid, int v)
{
    switch (oid)
    {
    case C_GEARUP:
        if (v == 0)
        {
            cout << "GearUp:" << v << endl;
        }
        break;
    }
}

Example 2:  Using the FLG_DISABLED flag

MkFsbusObject (BTP_D_IN, C_GEARUP, "GEARUP", EventCallback, 2, 33,
FLG_DISABLED);

    Enable(C_GEARUP);
```

MkUdpInterface

Syntax

```
FSUDP* MkUdpInterface (FSBUSUDPTYPE tp,
                       int port,
                       void(* cb)(FSUDP* p))
```

This function establishes a udp support.
After this function has been called, you can send data with UdpSend() or UdpSendTo().

A receiver service listens on the portnumber determined by the port argument. Any received datablock is passed to the callback function named in the 3rd parameter.

Parameters:

FSBUSUDPTYPE tp	UDP_RAW: Each received data block is passed without decoding to the callback function. UDP_FSBUS: Each received block is assumed to be a fsbus udp data block. The data is parsed and then passed to the callback function. Support begins with DLL version 2
int inport	The port number, to which the listener is bound to. The callback function will run, if a datagram on that port is received. If you don't want to receive data, set port to 0.
void(* cb)(FSUDP* p)	The pointer to the callback function, which receives data. You can obtain data and status information with a pointer to the associated FSUDP structure. This structure was created by the call to MkUdpInterface.

Return

A pointer to a FSUDP structure. This structure contains buffers and information about this udp interface. You need this pointer when using UdpSendTo.

MkTimer

Syntax

```
BOOL MkTimer (
    int oid,
    char* name,
    void(* cb)(int oid, int val, double dbl),
    DWORD tm,
    int flg)
```

This function creates a timer. Each timer generates a single or continuous events. The time is defined in milliseconds and uses the ordinary windows timing as a base.

oid	unique id of this object
name	optional user readable name of this object
cb	the callback function which is performed when the timer expires
tm	the time in ms until the timer expires
flg	if you specify FLG_ONESHOT, the timer is fired only one time. A 0 value will make a continuous firing timer.

Remarks:
This object is based on the standard win32 operating system timer. It's precision is not guaranteed to be accurate, because Windows is not a real time system. For more precise timing in ms range you should use the multimedia timers of windows.

You can modify a timer by SetTime function.

Return

Example

MkSound

Syntax

```
BOOL  MkSound (
        int oid,
        char* name,
        void(* cb)(int oid, int val, double dbl,
        char* soundfile)
```

The FSBUS DLL supports the DirectX mixer capabilities. A sound object is created with MkSound.

Return

Example

MkLoopSound

Syntax

```
BOOL  MkLoopSound (
          int oid,
          char* name,
          void(* cb)(int oid, int val, double dbl,
          char* soundfile)
```

The FSBUS DLL supports the DirectX mixer capabilities.

Return

Example

Pan

Syntax

```
void Pan(int oid, int pan)
```

This function controls a sound object indicated by an object id. The position between left and right speaker is defined by the pan value. The range is from 0 to 100.

Example

Play

Syntax

```
void Play(int oid)
```

This function controls a sound object indicated by an object id. The soundbuffer which was loaded and prepared at the object creation starts playing. If the sound object was played before, don't forget to call Rewind before you call Play again.

Example

Rewind

Syntax

```
void Rewind(int oid)
```

This function controls a sound object indicated by an object id. The position in the sound buffer is reset to beginning position.

Example

SetTime

Syntax

```
void SetTime(int oid, DWORD tm)
```

The SetTime function will modify the time of a previsously created FSBUS timer.
If you specify tm = 0, the timer is temporarily disabled until you call SetTimer again with a value greater 0.

Example

SetPower

(3.0.1)

Syntax

```
void SetPower(int powertype, int on)
```

The SetPower function will notify a specific powerstate. It controls all fsbus objects, which are optionally declared with a specific power-off value (FsbusPowerOffOptions()).

This function will loop through all objects, looking for a previously defined power-off value and in case of finding one and havng the same powertype, it will send either the actual value or the power off value, depending on the "on" parameter.

Example

```
Refer to FsbusPowerOffOptions()
```

Stop

Syntax

```
void Stop(int oid)
```

This function controls a sound object indicated by an object id. The sound stops playing.
If you call Play again, the soundbuffer is continued at the position where it was stopped before.
If the sound objects current position is at the end, don't forget to call Rewind before you call Play again.

Example

UdpDestination

Syntax

```
bool UdpDestination (FSUDP* udp, char* host, int port)
```

UdpDestination sets the internal variables, used by SendTo to values according to the host/port pair.

Host is either an ordinary internet address or an ip address with dotted decimals (89.1.1.2).
After the destination is set, UdpSend can be called numerous times.

UdpSend

Syntax

```
bool UdpSend(FSUDP* udp, unsigned char* buf, int count)
```

UdpSend will send a datablock to a specific port on a specific host.

Vol

Syntax

```
void Vol(int oid, int vol)
```

This function controls a sound object indicated by an object id. The volume is defined by the vol value. The range is from 0 to 100.

Example

FSBUS dataformat

This chapter describes the dataformat for FSBUS hardware controllers. As long as you are using the FSBUS DLL, you don't have to deal with these specifications.

The hardware parameters for serial communication is 19200bps, 8bit, no parity, 2 stop bit.

Data from FSBUS Router to Controller

The 5 C bit determine the CID. Up to 31 controllers can be addressed with 5 bit. There are 2 exceptions to the rule.
If C=0 then the sent data is interpreted by any controller (broadcast).
If C=31 then the dataframe is sent with an extended address format. This is a future solution for cockpits with more than 30 controller.

The 8 bit value R determines the command.

A value may have different length and format. A general format is shown in the table below. Value bits (V) are marked grey.

Byte	7	6	5	4	3	2	1	0
1	1	C_4	C_3	C_2	C_1	C_0	R_7	V_0
2	0	R_6	R_5	R_4	R_3	R_2	R_1	R_0
3	0	V_7	V_6	V_5	V_4	V_3	V_2	V_1
4	0	V_{14}	V_{13}	V_{12}	V_{11}	V_{10}	V_9	V_8
5	0	V_{21}	V_{20}	V_{19}	V_{18}	V_{17}	V_{16}	V_{15}
6	0	V_{28}	V_{27}	V_{26}	V_{25}	V_{24}	V_{23}	V_{22}

Data sent by IO Controller to FSBUS Router

The key controller of the IO board is organized by 1 to 8 rows, each with 8 bit corresponding to a switch.
The keycontroller requires a setup before use. That setup function defines the keytype of specific input lines.
If a change on any key is detected by a controller, an absolute key value is sent, or in case of rotaries a relative value is sent to FSBUS Router.

Byte	7	6	5	4	3	2	1	0
1	1	C_4	C_3	C_2	C_1	C_0	R_7	V_0
2	0	R_6	R_5	R_4	R_3	R_2	R_1	R_0
3	0	V_7	V_6	V_5	V_4	V_3	V_2	V_1

C0-4 CID of the controller, which sent this dataframe

R0-7 the number is calculated by row * 8 + bit, at which a change was detected
V0-7 a signed value of the key state

Common R-Commands

These are all commonly used to send a command from fsbus software to a controller. R-commands are numbered between 128 and 255.

R	Name	L	
128	Reset	2	0: Reset controller without displaying the CID 1: Reset controller and display the CID after reboot
129	SetCID	3	Parameter is the CID (Bit0-4), which is stored in eeprom. For safeteness, this command needs to be sent 3 times in sequence. Bit5-7 counts the sequence (0,1,2). If the CID is greater 30, a different format is used. The CID is a 7 bit wide integer, which occupies the sequence bits. Sequence bits are no longer in use for this format.
130	SetBright	3	Setup brightness 0 (dark) – 255(bright)
131	SetPower	3	This command simulates the battery state. A value from 0(battery empty) to 100(full) is supported. Usually the controllers will shutdown below a value of 80.
132	SetDecimalPoint	3	Position of decimal point. A value of 0 turns off the decimal point. 1 is the right most position.
133	SetBaseBright	3	Setup the controller individual brightness value, stored in internally eeprom.
134	SetCIDExt	3	Parameter is the new CID, to which the controller will respond. For safeteness, this command needs to be sent 3 times in sequence.

R-Commands for IO Controller

Except the display format, any IO controller uses this multi purpose data format.

R	Name	L	to fsbus router	to controller
0-7	Key R0 bit 0-7	3	Keyvalue	setup keytype
8-15	Key R1	3	Keyvalue	setup keytype
16-23	Key R2	3	Keyvalue	setup keytype
24-31	Key R3	3	Keyvalue	setup keytype
32-39	Key R4	3	Keyvalue	setup keytype
40-47	Key R5	3	Keyvalue	setup keytype
48-55	Key R6	3	Keyvalue	setup keytype
56-63	Key R7	3	Keyvalue	setup keytype
72-79	A-In	3	analogue value 0-7	tolerance of analogue input
80-87	A Out 0-7	3		value of analogue out
88-95	D-Out 0: 0-7	2		value of a digital output bit on port 0
96-103	D-Out 1: 0-7	2		value of a digital output bit on port 1
104-111	D-Out 2: 0-7	2		value of a digital output bit on port 2
112-119	D-Out 3: 0-7	2		value of a digital output bit on port 3
120	D-Outbyte 0	3		value of a digital output port 0
121	D-Outbyte 1	3		value of a digital output port 1
122	D-Outbyte 2	3		value of a digital output port 2
123	D-Outbyte 3	3		value of a digital output port 3
124	A-In Mask	3	mask of analogue input pins in use.	mask of analogue input pins in use.
125	A-Out Mask	3	mask of analogue output pins in use	mask of analogue output pins in use
128-135	A-Tolerance	3	tolerance of analogue input	
136-155	DDP area	3	Device dependent parameters	Device dependent setup parameters
200-207	D-Out 4: 0-	2		value of a digital output bit

	7			on port 4
208-215	D-Out 5: 0-7	2		value of a digital output bit on port 5
216-223	D-Out 6: 0-7	2		value of a digital output bit on port 6
224-231	D-Out 7: 0-7	2		value of a digital output bit on port 7
232	D-Outbyte 4	3		value of a digital output port 4
233	D-Outbyte 5	3		value of a digital output port 5
234	D-Outbyte 6	3		value of a digital output port 6
235	D-Outbyte 7	3		value of a digital output port 7
255	Init	3	The controller was new startet. Parameter is the version number	

Display Controller

the display controllers display values are in a special format. It is detected by a 0 in bit 1 of byte 0.
Each segment value is transmitted as a 4bit nibble. The dataformat has variable length. The final byte is detected by a 1 in bit 6 of the last byte.

Byte	7	6	5	4	3	2	1	0
0	1	C_4	C_3	C_2	C_1	C_0	**0**	x
1	0	0	A_3	A_2	B_3	B_2	B_1	B_0
2	0	0	A_1	A_0	C_3	C_2	C_1	C_0
3	0	0	D_3	D_2	E_3	E_2	E_1	E_0
4	0	**1**	D_1	D_0	F_3	F_2	F_1	F_0

C0-4 Controller ID
A,B,C,D,E,F nibbles for max. 6 segments

The order of the segments is from left to right: F E D C B A

The decimal point is defined by a R-Command 132. The value defines the position of decimal point:

F	**E**	**D**	**C**	**B**	**A**
6	5	4	3	2	1

One nibble (4Bit) can show 16 different characters:

Wert	Anzeige	Wert	Anzeige
0	0	8	8
1	1	9	9
2	2	A	-
3	3	B	S
4	4	C	t
5	5	D	d
6	6	E	E
7	7	F	

More than 30 controllers

Some years ago, when fsbus was designed, a maximum of 31 controllers were allowed. Now this barriere is enlarged to use up to 99 controllers.

CID 31 is now reserved for longer CID values. In this case there is an extra byte following Byte 1. This byte covers the range from 31 – 99.

Byte	7	6	5	4	3	2	1	0
1	**1**	**1**	**1**	**1**	**1**	**1**	R_7	V_0
2	**0**	C_6	C_5	C_4	C_3	C_2	C_1	C_0
3	**0**	R_6	R_5	R_4	R_3	R_2	R_1	R_0
4	0	V_7	V_6	V_5	V_4	V_3	V_2	V_1
5	0	V_{14}	V_{13}	V_{12}	V_{11}	V_{10}	V_9	V_8
6	0	V_{21}	V_{20}	V_{19}	V_{18}	V_{17}	V_{16}	V_{15}
7	0	V_{28}	V_{27}	V_{26}	V_{25}	V_{24}	V_{23}	V_{22}

The extended format is used automatically, if you specify CID’s greater 30.
Please remember:
The usage of this extended format requires new firmware in the controllers.

Upgraded firmware for the controllers will be available for the display controllers in January 2009. Older firmware can still be used with CID’s in range 1 – 30 without any change.

UDP dataformat for software cockpits

This chapter describes the dataformat used to supply the fsbus software gauges. This software is not part of the fsbus project, but you can use the data for your own.

The software sends the required data automatically on port 28150 and the same data on port 28151. Sending data twice on different ports allows an additional second client application on the same machine.

Data begins with a 2 byte magic number of **0xdafb** and is followed by 2 byte sequence number.
Next is a variable number of data pairs, consisting of a **2byte id** and a **4 byte float value or 4 byte integer**. Maximum number of pairs is **100**;
Integer values are marked with "Integer".

ID	Description	Min	max
0	Set the current gauge to move and size 0 IAS 1 Attimeter 2 Altimeter 3 Turn Coordinator 4 HSI 5 Vertical Speed 6 RMI 10 Manifold press L 11 Manifold press R 12 RPM L 13 RPM R 14 FF L 15 FF R 16 Oil L 17 Oil R 18 Temp L 19 Temp R	0	6
1	X delta position of the current gauge		
2	Y delta position of the current gauge		
3	Scale factor of the current gauge		
4	Base light This value is added (subtracted) from the daylight value. It will control brightness of the display		
5	Assign a model number to the current gauge. Disable a gauge by value -1		
10	Day/Night	0 (night)	100 (day)
11	Light	0 (off)	1(off)
16	Indicated Air Speed	0	1000
17	Barberpole value	0	1000
18	Variable speed scale		

	(Beech Baron only)			
32	Attitude Pitch	-50	50	
33	Attitude Bank	-50	50	
34	Attitude FD Vertical	-50	50	
35	Attitude FD Horizontal	-50	50	
36	Attitude Center	0	100	
48	Altitude			
49	Air pressure			
64	Turn Coordinator Plane			
65	Turn Coordinator Ball			
80	HSI Compass			
81	HSI Nav Flag			
82	HSI Heading Flag			
83	HSI To Flag			
84	HSI Nav Direction			
85	HSI Nav Hor Needle			
86	HSI Nav Vertical Needle			
87	HSI Heading Bug			
96	Vertical Speed	-2	+2	
112	RMI Compass	0	360	
113	RMI NAV Needle	0	360	
114	RMI ADF Needle	0	360	
115	Instrument Air	2.5	6.5	
132	Left Fuel Level %	0	100	
133	Right Fuel Level %	0	100	
148	Main Bus Voltage	0	28	
149	Alternator1 Load			
150	Alternator2 Load			
192	Manifold Pressure Left	0	40	
196	Manifold Pressure Right	0	40	
200	RPM Left	0	35	
204	RPM Right	0	35	
208	Fuel Flow Left	0	30	
212	Fuel Flow Right	0	30	
216	Oil Temp Left	0	120	
217	Oil PSI Left	0	120	
220	Oil Temp Right	0	120	
221	Oil PSI Right	0	120	
224	Temp CHT Left	0	120	
225	Temp EGT Left	0	120	
228	Temp CHT Right	0	120	
229	Temp EGT Right	0	120	
256	NAV1 freq	Integer		
257	NAV1 stdby	Integer		
258	NAV2 freq	Integer		
259	NAV2 stdby	Integer		
260	COM1 freq	Integer		
261	COM1 stdby	Integer		
262	COM2 freq	Integer		
263	COM2 stdby	Integer		
264	ADF	Integer		
265	Transponder	Integer	A transponder value has 4 digits. A leading 5th digit is used to show a cursor. 1: cursor on digit0 (rightmost)	
266				
267	AP Altitude	Integer		

273	AP Vertical Speed	Integer
270	AP Display Flags	Integer 2^0 HDG 2^1 ALT 2^2 NAV 2^3 VS 2^4 APR 2^5 AP 2^6 HDG 2^7 NAV 2^8 APR 2^9 REV 2^10 ALT
271	DME NM	Integer
272	DME KT	Integer
273	AP Vertical Speed	Integer

A display can contain up to 25 gauges. The gauges are numbered from 0 to 24. Each gauge has an associated gauge model. The model numbers normaly are the same as the gauge numbers, but can have an individual associaton.

Since you can move and size each gauge individual, the number is only an ordering scheme without any functional importance.

Default association of the gauges. You can assign a different model to each gauge or disable it by assigning the special value -1.

Gauge	Model	
0	0	IAS indicator
1	1	Attitude indicator
2	2	Altimeter
3	3	Turn Coordinator
4	4	HSI
5	5	Vertical Speed
6	6	RMI
7	7	Instrument Air
10	10	Manifold Pressure
11	10	Manifold Pressure
12	12	RPM
13	12	RPM
14	14	FF
15	14	FF
16	16	Oil Temp/PSI
17	16	Oil Temp/PSI
18	18	Temp CHT/EGT
19	18	Temp CHT/EGT
30	30	NAV1
31	30	NAV2
32	30	COM1
33	30	COM2
34	34	ADF
35	35	Transponder
36	36	Autopilot