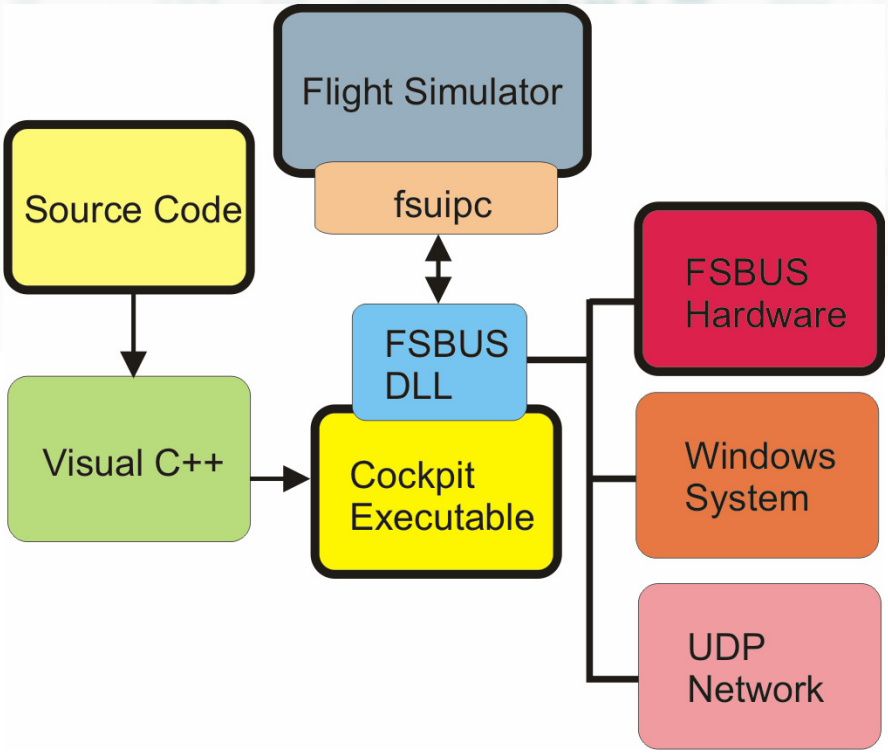
FSBUS

DLL Manual

Dirk Anderseck 2009



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.

Changes

11/2009 fbus.dll Version3 is now in pure C-style to support the Borland compilers. There are 2 functions affected

```
BOOL FsbusWrite(unsigned char* buf, int count);
Is replaced by
BOOL FsbusWriteRaw(unsigned char* buf, int count);
```

The function

```
BOOL FsWrite (int oid, int i32);
BOOL FsWrite (int oid, double d);
BOOL FsWrite (int oid, __int64 i64);
BOOL FsWrite (int oid, UVAR u);

Is replaced by

BOOL FsWriteInt (int oid, int i32);
BOOL FsWriteDbl (int oid, double d);
BOOL FsWriteInt64 (int oid, __int64 i64);
BOOL FsWriteUnion (int oid, UVAR u);
```

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 room for adding so called offsets of 3rd party software.

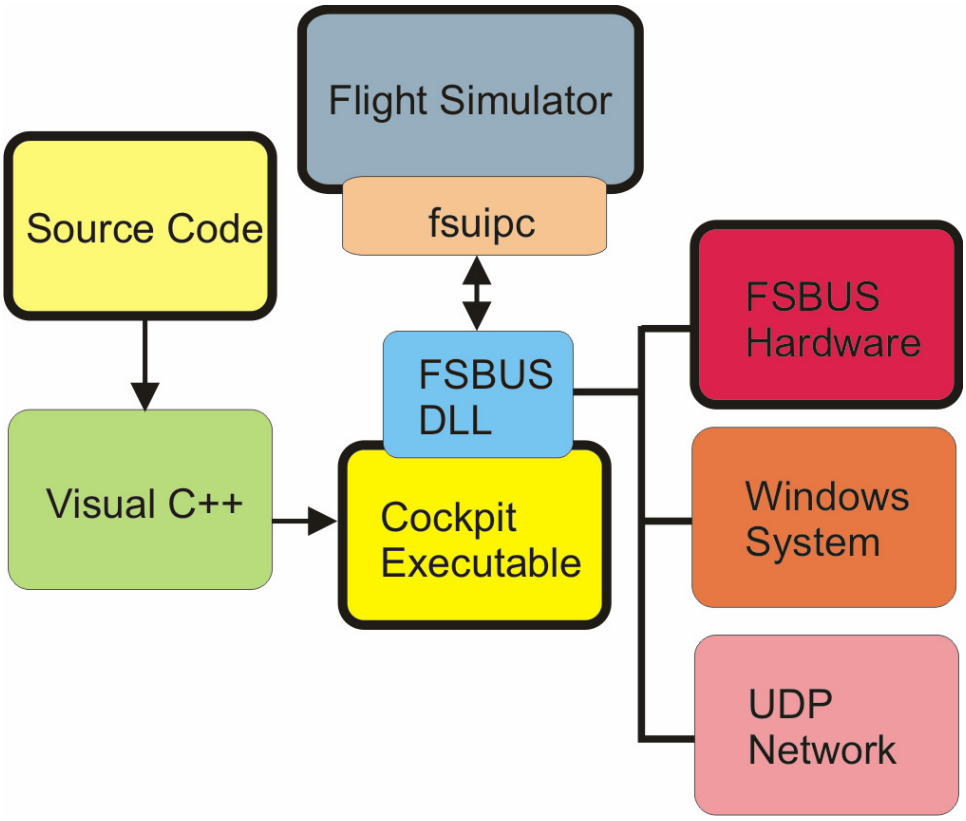
I have designed the interface as easy as possible. That's the reason why the syntax is classic C-style for console applications.

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 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.

Architecture

The FSBUS DLL covers many aspects of a Flightsimulator Cockpit Control. It has been made to control the fsuipc as well as the FSBUS hardware. You will have to write some C code and compile it into an executable, which then acts as your cockpit software.
Using a C compiler will generate huge programs at highest perfomance.



The idea behind the software is based on objects, which you create according to your needs. Each hardware or flightsim item will be such an object. The code to create the objects is one of your programming tasks.

Each object passes Data, which is sent to it from the hardware or from flightsim, to a callback function. This is the second programming task you will write. Objects fire events, when a value changes or a key is switched. You react on such events in the callback code.

That in generally is the FSBUS DLL concept. The timing, the sometimes complex interfaces is all handled inside the DLL. Your code, written in a powerfull language like C, compiled and linked with the Microsoft C-Compiler will produce high end solutions running at highest possible speed.

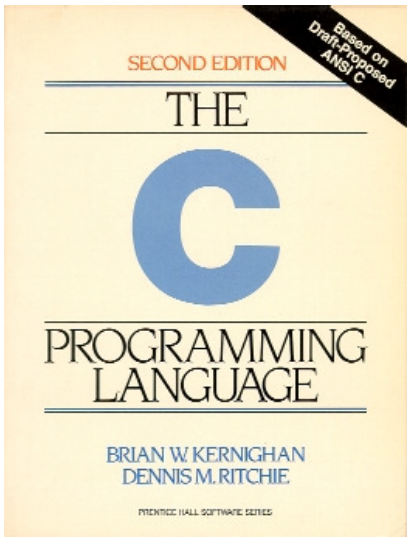
Programming skills

I you are C programmer, you can start with the example code. I do not expect any hurdles, you cannot jump over.

If you are programmer but don't have experience with C, i recommend one of the many introductions to C which are available in the internet or in a bookstore.

For all others, please don't fear C. There is no need to learn the much more complex C++. You can if you like, but there is not a single line you will not understand with basic C knowledge. To get into C, take a good education book or something else. It will be as easy as Visual Basic. Finally, practice is the way to get ready for your cockpit application.

My book recommendation is the good old White Bible.



Written by Brian Kernighan and Dennis Ritchie, who originally designed and implemented the language.

Files

There are some files coming with the FSBUS DLL.

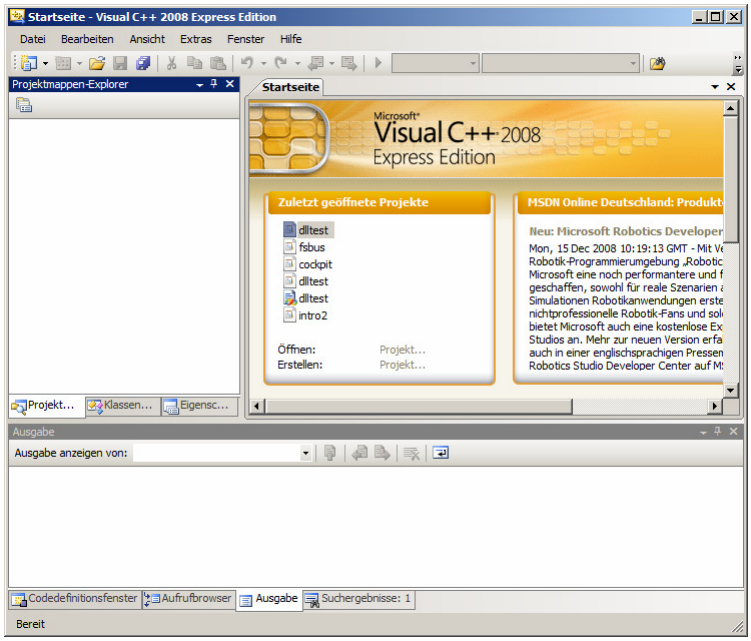
fsbus.dll	Please copy this DLL into your applications binary directory. It is loaded, when your exe program starts.
fsbus.lib	This file is the interface to the DLL. You will have to add this file to your project.
fsbus.h	contains all function and dataformat definitions of the DLL. You will have to include it in any of your C-programs, which use the fsbus dll.

There are also some PDF files containing documentation.

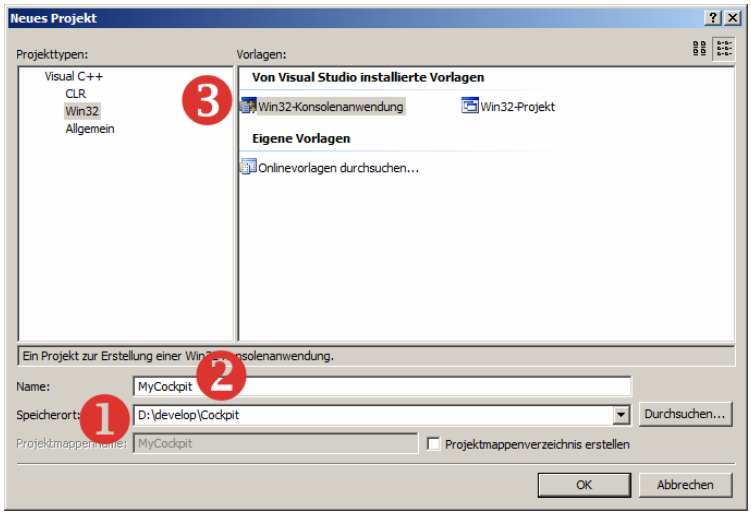
To develop your own cockpit software, you may use the Microsoft Visual C++ Express Edition. You can download that compiler from the Microsoft homepage for free.
If you are using some Windows functions (i am sure you will) don't forget to download the Windows SDK as well.

Make your own cockpit software

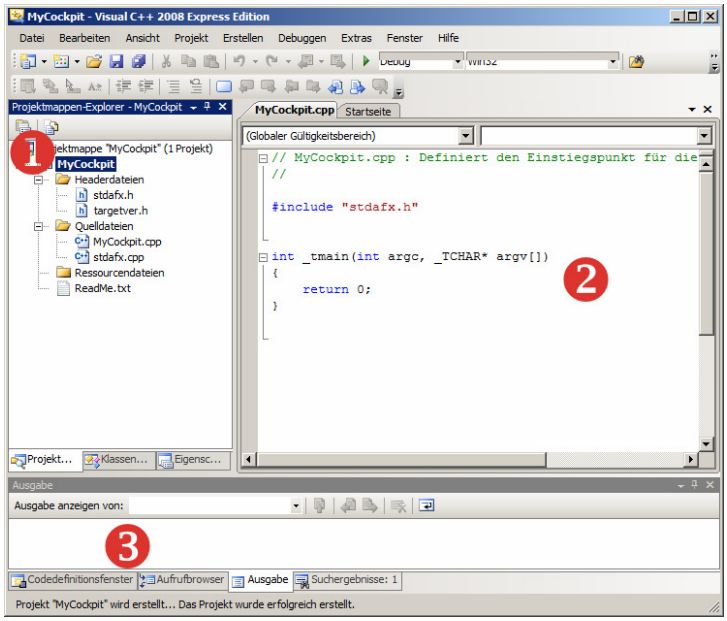
This introduction shows how to start a software project with Visual C++ Express Edition 2008.



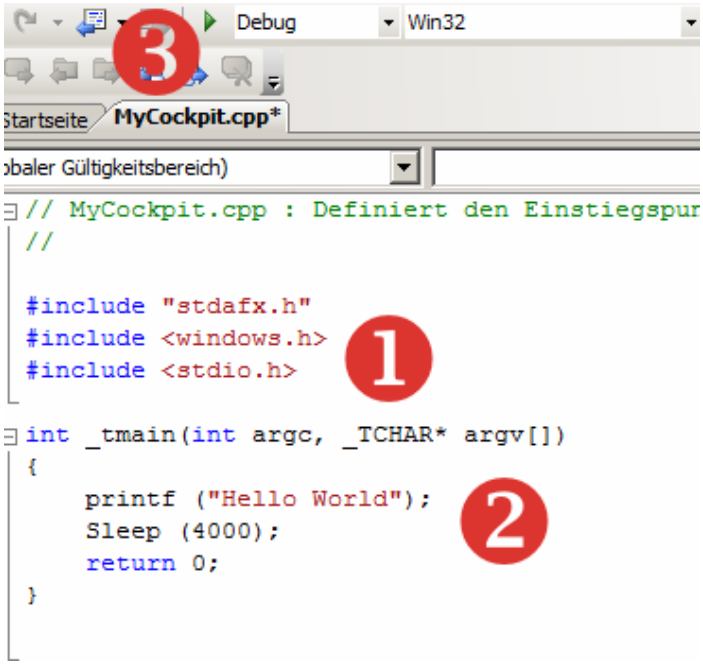
After starting the development plattform, select [File] [New] [Project] in the menu



1 Select a directory, where you like to store all project files.
2 Select a name for your cockpit software.
3 Determine the type of your project. If you are beginner, i strongly recommend Win32 Console Application.



The wizzard now produces some files.
1 This is the list of all files in the project
2 The main file mycockpit.cpp contains the function _tmain, which doesn't do anything so far.
3 A message area, which shows all errors and warnings. You will see it empty for now. Believe me, it will be the first and last time seeing this empty!

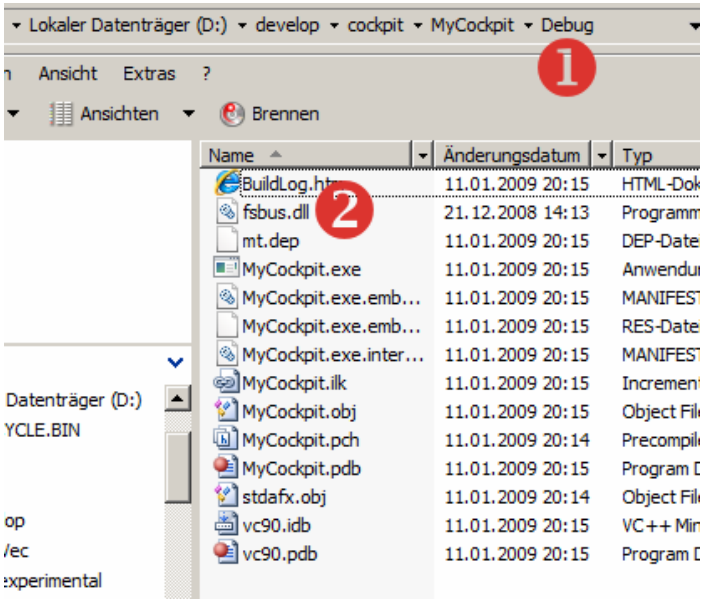


Now add a few lines into the code, to get the famous Hello World example running.

1 the two include statements adds two important files with a lot of declarations concerning the windows environment into the project. These files are part of the compiler and the Windows SDK, which you must download from the microsoft homepage.

2 the printf shows a text on the console and the Sleep command waits four seconds, after the console disappears because the program is finished now.

3 the green arrow starts the compiler and linker. If anything is ok, it will then start the programm.

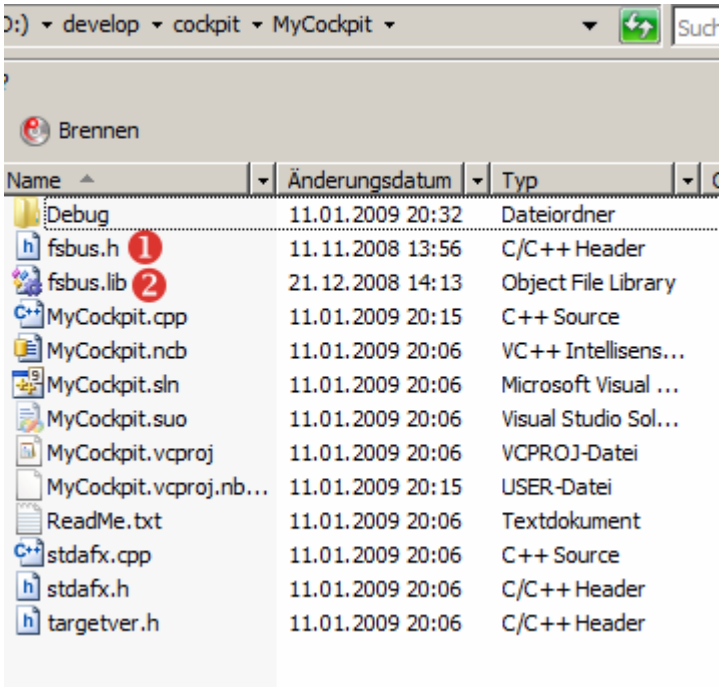


Now it's time to add the fsbus dll into the project.

1 the MyCockpit directory contains a Debug directory, which was created automatically.

2 I recommend to copy the fsbus dll file into this directory.

Later on when the development is finished you may copy it into the windows system directories.

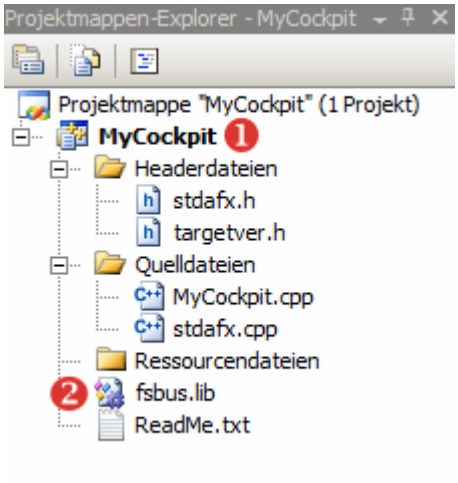


The mycockpit directory contains all source files of your cockpit project.

Please copy the fsbus.h and fsbus.lib into this directory.

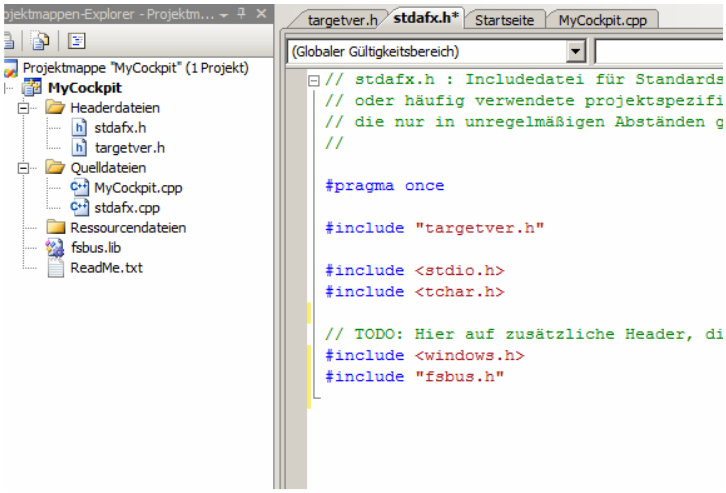
1 the fsbus.h file contains definitions of the functions and variables in the dll file.

2 the fsbus.lib file contains informations for the linker.



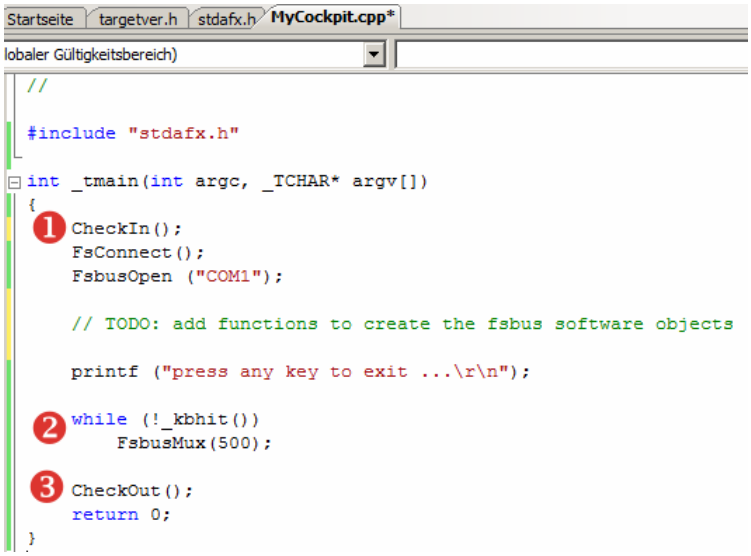
You must import the fsbus.lib file (NOT THE DLL) into your project.
1 Rightclick MyCockpit and then [Add] [existing element ...], select the .lib file and press Enter.

2 the fsbus.lib is now linked into your software and is the key to access all fsbus.dll functions in your software.



Now we are going to prepare the project for large cockpit projects.

Stdafx.h is a good place to add the required .h files. Please add any .h file below the include "fsbus.h" line.



Next step is the main body of the programm. This body may look similar in any of the cockpit projects.

1 CheckIn is alway the first DLL function, you have to perform. It initializes all parts of fsbus. FsConnect will open a connection to the still running flight simulator. FsbusOpen connects to the fsbus hardware.

2 FsbusMux is the one and only function which keeps the ball playing. It is performed as long as a key is pressed.

3 After a key pressure, the CheckOut finishes fsbus and the process exits.

Up to this step, the project is prepared to run fsbus. You may save the complete path and it can be used as a base for many cockpit projects.

Now we will implement all the individual objects which makes a cockpit built with fsbus hardware running with the flight simulator. Each switch, button display, motor and so on needs to be declared as a software object. Each flightsim variable, which is described in fsuipc documentation needs a declaration as a software object as well.

If we think about the number of elements in a real cockpit, we will easy count thousands of objects. Any project will result in chaos code, if we put any command into one file.

A recommended way is to group the objects in a throttle group, a yoke group and so on. Let's discuss the object names and object id's.

Each object, fsbus and flightsim, has an id. This id is provided by the programmer and has to be unique. It is called OID (object id). The total number of objects must not exceed 4096. This is due to performance issues.

The programm language C makes it easy to assign symbolic names to numeric id's and a program is more readable by using names instead of id's.

The OID is devided in a group part and an object in a group.

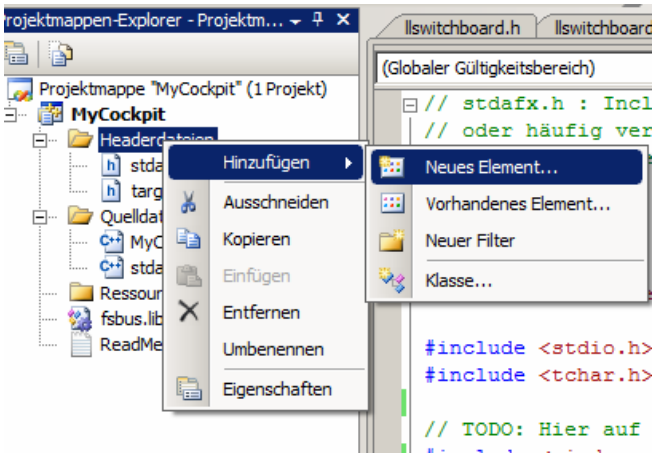
11	10	9	8	7	6	5	4	3	2	1	0

With 12 bit we can address 4096 different objects. Bits 0-4 allows us to define 32 objects per group.
Bits 5-11 allows to build 128 groups.

The process of numbering the objects is as follows:

- Build a group of up to 32 flightsim and fsbus elements
- Define a unique group number in stdafx.h
- Enumerate all elements in the corresponding .h file
- Add a function which creates all objects in the corresponding .cpp file
- Add a function to handle all the events in the corresponding .cpp file

Let's make our own new control group named "switchboard". Any group needs 2 new files: switchboard.cpp and switchboard.h

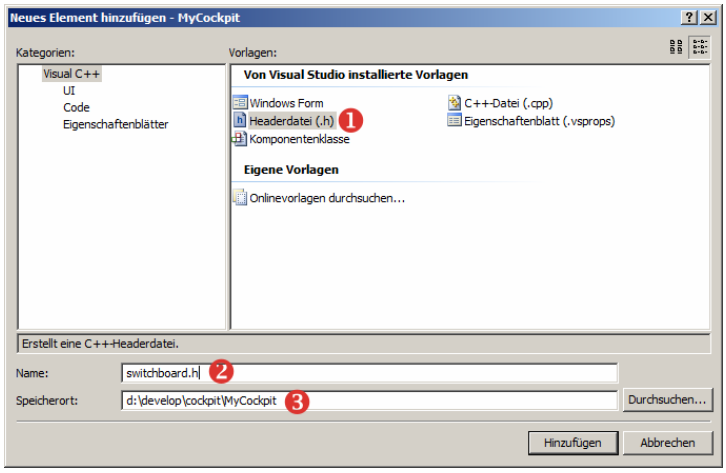


The headerfiles are extended by a new element called switchboard.h .

1 Mark filetype as headerfile.

2 type the name of the new file

3 put it into the project folder




```
#ifndef __SWITCHBOARD_H
#define __SWITCHBOARD_H

void BuildSwitchboardObjects ();
void cbSwitchboard (int oid, int val, double dval);

enum e_switchboard {
    C_SWITCH_LANDINGLIGHT_L = OID_LLSWITCHBOARD_GROUP,
    C_SWITCH_LANDINGLIGHT_R,
    C_SWITCH_STROBE,
    FS_LIGHTS
};

#define FS_LIGHTS_NAV       0x01
#define FS_LIGHTS_BEACON    0x02
#define FS_LIGHTS_LANDING   0x04
#define FS_LIGHTS_TAXI      0x08
#define FS_LIGHTS_STROBES   0x10
#define FS_LIGHTS_PANEL     0x20
#define FS_LIGHTS_RECOGNITION   0x40
#define FS_LIGHTS_WING      0x80
#define FS_LIGHTS_LOGO      0x100

#endif
```

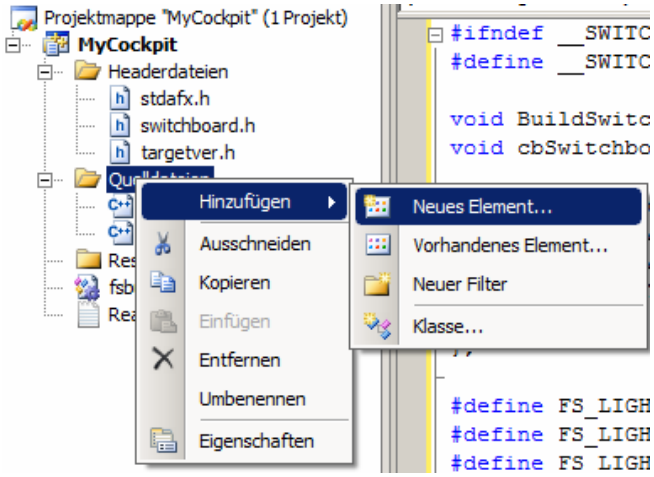
Now you can fill the includefile with your individual cockpit definitions.

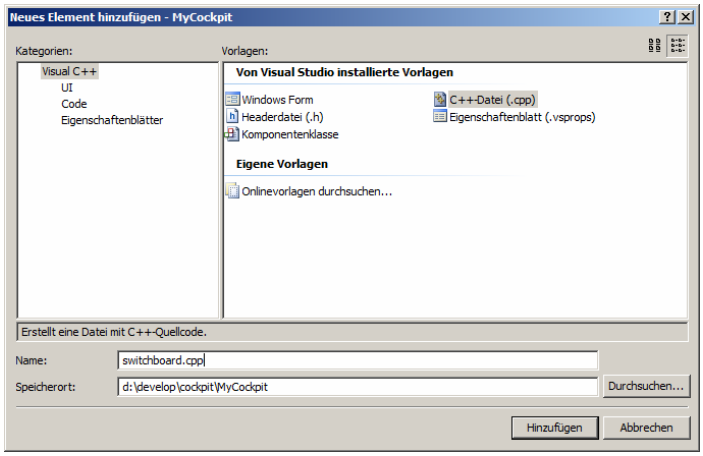
1 this is to prevent multiple include in your code. You will see this in nearly any include file.

2 there are two functions, which are declared here.

3 in the file stdafx.h, the object group id OID__LLSWITCHBOARD is defined. Here are all objects of this group enumerated and the first is assigned the first id in the group. Any further object gets a number plus one. Do not put more than 32 objects into one group.

4 A look into the fsuipc doc shows the light bits, which are given a name here.






```
#include "stdafx.h"                                                    1
static int Lights;                                                     2

void BuildLLSwitchboardObjects ()                                      3
{
    MkFsbusObject (BTP_D_IN, C_SWITCH_LANDINGLIGHT_L,   "",cbLLSwitchboard,2, 51);
    MkFsbusObject (BTP_D_IN, C_SWITCH_LANDINGLIGHT_R,   "",cbLLSwitchboard, 2, 52);
    MkFsbusObject (BTP_D_IN, C_SWITCH_STROBE,   "",cbLLSwitchboard,2, 54);
    MkFsObject(FS_LIGHTS,   "",cbLLSwitchboard, 0x0D0C, 2,      TP_I16, FS_NORMAL);
}

void cbLLSwitchboard (int oid, int val, double dval)                   4
{
    switch (oid)
    {
    case C_SWITCH_STROBE:                                              5
        if (val)
            Lights |= FS_LIGHTS_STROBES;
        else
            Lights &= ~FS_LIGHTS_STROBES;
        break;
    case FS_LIGHTS:                                                    6
        Lights = val;
        return;
    default:
        return;
    }
    FsWrite (FS_LIGHTS, Lights);
}
```

! Version 3 of the FSBUS DLL is now in C-style to support other compilers (Borland). The function FsWrite
which had different arguments for different data types has now different names as well. In this example FsWriteInt(FS_LIGHTS,LIGHTS) is required.

Making a new source file is the same procedure as before.

Now the source code.

1 the file stdafx.h, which contains all include files is included here.

2 a static variable with the name Lights is defined.

3 the function BuildLLSwitchboardObjects will be called from the main function. It creates all Fsbus and flightsim objects.

4 this function is called from inside the DLL at every time, a value changed.

5 if the strobe switch is toggled, the code below will handle this event and set or reset a bit in the Lights variable.
At the bottom of the code you see FsWrite, the function to send a value to the flightsim.

6 this code is performed, when flightsim changes the lights by pressing a software knob or selecting a new plane. This also modifies the Lights variable.

```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    CheckIn();
    FsConnect();
    FsbusOpen ("COM1");

    // TODO: add functions to create the fsbus software objects
    BuildLLSwitchboardObjects ();  1

    printf ("press any key to exit ...\r\n");

    while (!_kbhit())
        FsbusMux(500);

    CheckOut();
    return 0;
}
```

Finally 2 more files need some new code.
The cockpit .cpp with its main functions calls the build function (1)

```
#pragma once

#include "targetver.h"

#include <stdio.h>
#include <tchar.h>

// TODO: Hier auf zusätzliche Header, die das Programm erforde
#include <windows.h>
#include "fsbus.h"

#define OID_LLSWITCHBOARD_GROUP     (1 << OID_CONTROL_BITS)  1

#include "switchboard.h"  2
```

The stdafx file:

1 the group gets a unique id and a name.

2 the groups include file is added

How to define the Flightsim Objects

All flightsim variables are accessed through the fsuipc interface. In order to use it, each one needs to be created by your software. This is done by the function MkFsObject. It is mandatory to declare a unique object id, the fsuipc object, the length of the fsuipc variable according to the fsuipc developer manual and the data type, which is used to make a conversion before the value is passed to any of your callback functions.
The callback function needs to be declared, if the value read from fsuipc should be passed to your code.

```
MkFsObject(
        FS_ENGINE1_THROTTLE_CONTROL,  // the unique object id
        "ENGINE1_THROTTLE_CONTROL",   // the optional user readable name of the object
        cbThrottle,                    // the callback function
        0x089A,                              // the fsuipc offset
        2,                             // number of bytes in fsuipc
        TP_I16,                   // data type Integer 16 bit
        FS_NONE);                   // do not poll(read) this value
```

Defining Fsbus Objects

All FSBUS hardware parts needs to be created as an object by your software. This is done by the function MkFsbusObject. It is mandatory to declare a type, a unique object id, the controller id and register id of the hardware object.
The callback function needs to be declared, if the value read from fsbus should be passed to your code.

```
MkFsbusObject (
        BTP_A_IN,                       // type of the fsbus control
        C_THROTTLE_LEVER_LEFT,          // unique object id
        "THROTTLE_LEVER_LEFT",          // the optional user readable name
        cbThrottle,                     // the callback function
        2,                              // Controller ID of FSBUS card
        70);                            // Register ID of this lever
```

How to exchange data with Flightsim

With these predefined objects you have access to any of the thousands of parameters inside Flightsim. To access an object, you pass the object ID (oid) to one the FSBUS DLL functions.

There are two strategies to read a value from flightsim, POLLING and DIRECTREAD. All frequently used values should be polled automatically by the internal FSBUS DLL functions.

POLLING of a specific variable is enabled by setting the poll parameter in fsobjects.cpp to one of the constants FS_NORMAL, FS_LAZY or FS_QUICK.
If you set this parameter to FS_NONE, polling is disabled.

Anyway you can obtain a value from flightsim by calling FsReadDirect(). This should be used for parameters read once or very few times. Another good choice of FsReadDirect is, if you read variables with special length or format.

How to obtain a value by polling

Each change of the value is sent as an event to your event callback function. There are 3 parameters in the callback, the first identifies the source, the second one contains a converted integer value and the third is the floating point value, which is valid for fsuipc parameters in 8 byte floating point format.

Example: display a message when the pause status of flightsim changes

```
// create the flightsim object
 MkFsObject(FS_PAUSEINDICATOR, "PAUSEINDICATOR", EventCallback, 0x0264,
                  2, TP_UI16, FS_NORMAL);

// code for event handling
void EventCallback (int oid, int value, double dval)
{
    switch (oid)
    {
    case FS_PAUSEINDICATOR:
        if (value == 0)
            printf ("continue flying\r\n");
        else
            printf ("time for coffee\r\n");
        break;
    }
}
```

Example: read the objects read buffer, instead of using the converted int "val".

```
void EventCallback (int oid, int val, double dval)
{
    UVAR *newvar;
    UVAR *lastvar;

    switch (oid)
    {
    case FS_PAUSEINDICATOR:
        FsGetCompareBuffers (oid, &newvar, &lastvar);

        if (newvar->urd.i16 == 0)
            printf ("continue flying\r\n");
        else
            printf ("time for coffee \r\n");
        break;
    }
}
```

Please look at FsGetCompareBuffers(). You may wonder about a second variable pointer lastvar. This allows not only to obtain current values, but also the value at last polltime. Now you can compare two polled values and detect any difference.

This is very usefull for values with a high resolution as the altitude value which changes very frequently, because the resolution is in some millimeters.

It makes no sense to perform actions on a millimeter base. It only costs cpu time.

How to obtain a value without polling

The above method of polling values is the most effective way for fsuipc. It should be used for frequently read variables.

Variables, read only few times can be read by FsReadDirect, which obtains the value immediatly.

Example:

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

The arguments for this function are the fsuipc based offset, the length information and an address of a variable, where to put the read value.

How to exchange data with Fsbus hardware

You need to declare and instanciate each FSBUS object with the function

```
void MkFsbusObject(FSBUSTYPE ftp, int oid, char* n, int c, int r, int flg =
0)
```

The parameters are:

ftp The type of hardware. The predefined id's are:

```
BTP_D_IN    Digital Input (keys, buttons, sensors)
BTP_ROTARY  Rotary switches
BTP_A_IN    Analogue input
BTP_D_OUT   Digital output
BTP_DISPLAY 7 segment display
BTP_A_OUT   Analogue output
```

oid The id of this object (1-3999). You may choose your own numbering scheme.

name Name of this control or NULL if you want to leave it anonymous.

c Controller ID

r Register in the controller

flg Flags

```
FLG_DISABLED
FLG_ONESHOT
```

For each object a unique id is needed.

The easy way is to define it and apply the id's by yourself.
Example:

#define C_ADF1ROTARY 1001
#define C_NAV1DISPLAY 1002

The more complex a cockpit grows, it makes sense to group the id's as described before.

Each hardware event (button, switch, rotary, pot) is sent as an event to your event callback function. There is one parameter in the callback, that contains the converted integer value.

There are some functions to access the FBBUS hardware:

BOOL FsbusWrite(int oid, int val) Send a value to an object with

	an object id . The object must be created before use.
BOOL FsbusWrite(unsigned char* buf, int count)	You can build your own buffer according to the fsbus software documentation and send this raw buffer.
BOOL FsbusWriteFmt2(int cid, int rid, int val)	FSBUS software format has 3 different length of command blocks. You can send a 2 byte long format with this function.
BOOL FsbusWriteFmt3(int cid, int rid, int val)	FSBUS software format has 3 different length of command blocks. You can send a 3 byte long format with this function.
BOOL FsbusWriteFmtVar(int cid, int rid, int val)	FSBUS software format has 3 different length of command blocks. You can send a variable length format with this function.

Summary of files contents

The following table shows the files of a small project and its contents.

stdafx.h

```
#pragma once

#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include "fsbus.h"

#define OID_GAUGES_GROUP            (1 << OID_CONTROL_BITS)
#include "gauges.h"
```

gauges.h

```
#ifndef __GAUGES_H
#define __GAUGES_H

typedef enum {
      C_AIRPRESSROTARY = OID_GAUGES_GROUP,
      C_AIRPRESSDISPLAY
} EGAUGE;

void BuildGaugesObjects();
void cbGauges (int oid, int val, double dval);

#endif
```

gauges.cpp

```
#include "stdafx.h"

void BuildGaugesObjects ()
{
    MkFsbusObject (
        BTP_ROTARY, C_AIRPRESSROTARY,
        "",cbGauges,  2, 6);

    MkFsbusObject (
        BTP_DISPLAY, C_AIRPRESSDISPLAY,
        "",cbGauges,  12, 0);
}

void cbGauges (int oid, int val, double dval)
{
  double dbl;
  int x;

  switch (oid)
  {
    case FS_ALTIMETERPRESSURE:
      dbl = (double)val / 16.0 * 2.953;
      FsbusWrite (C_AIRPRESSDISPLAY, dbl / 16);
      break;
  }
}
```

cockpit.cpp

```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
```

```
    CheckIn();
    BuildGaugesObjects();
    FsbusOpen("COM5");
    printf ("FSBUS hardware connected ...\r\n");
    FsConnect();
    printf ("Flightsim connected ...\r\n");
    printf ("press any key to exit ...\r\n");
    while (!_kbhit())
      FsbusMux(500);
    CheckOut();
}
```

Error Handling

There are 3 different modes to deal with errors. The error mode is defined by the internal variable "ErrorMode" which is of type ERRORMODE. It is declared in fsbus.h.

The possible values are EM_STOPEXECUTION (default), EM_RESUMERETURN, EM_THROWEXCEPTION.

The default (EM_STOPEXECUTION) will show an error message on the console and after a keypress, the application exits. This suites the needs of most applications.

If you like to handle all errors by yourself, you can set ErrorMode = EM_RESUMERETURN. In this case, the error text is not shown on console, but copied into the variable ErrorText. The function which caused the failure will return with FALSE (if it is one of the BOOL ... functions) and you can code your own error handling.

Own error handling may be usefull, if you want to control the initialization phase. One example might be waiting for the flightsim until it has started.

Example code:

```
ErrorMode =  EM_RESUMERETURN;

int retry;

for (retry=0; retry < 5; retry ++)
{
        If (FsConnect() == TRUE)
                break;
        Sleep (5000);
}

if (retry >= 5)
{
     printf ("cannot open flightsim\r\n");
     exit (1);
}
ErrorMode = EM_STOPEXECUTION; // continue with default error mode
```

People writing C++ application frequently use a throw/catch mode to handle errors. If you want the DLL to throw an exception, set ErrorMode = EM_THROWEXCEPTION. On error it will throw an exception with one argument of type LPSTR.

License

FSBUS DLL is free for home users. To support further development, professional users or companies may buy a license to use FSBUS.DLL in commercial environment.

Please copy the license file into the directory, where the fsbus.dll resides.

```
FSBUS.INI

[LICENSE]
      NAME = My company
      LICENSE = A4 D3 XF AM B5
```

There is no functional difference between a licensed DLL and the free edition. Only the startmessage on FS will show the company name declared as Name in the fsbus.ini file.

Running your cockpit application on a PC without having Visual Studio installed

The Microsoft Visual C++ 2008 Redistributable Package (x86) installs runtime components of Visual C++ Libraries required to run applications developed with Visual C++ on a computer that does not have Visual C++ 2008 installed.

This package installs runtime components of C Runtime (CRT), Standard C++, ATL, MFC, OpenMP and MSDIA libraries. For libraries that support side-by-side deployment model (CRT, SCL, ATL, MFC, OpenMP) they are installed into the native assembly cache, also called WinSxS folder, on versions of Windows operating system that support side-by-side assemblies.

You can download the Redistributable Package from the Microsoft Website.