IN THE PIPE, FIVE BY FIVE

A DAY BY DAY LOOK INTO THE DEVELOPMENT OF FLIGHT SIMULATOR ADD-ONS AND XNA GAME DEVELOPMENT

WEDNESDAY, JANUARY 24, 2007

A Ten-Thousand Foot View of SimConnect - Part 4

Well, I'm back.

With the holidays over and my job situation stabilized for the moment (I'm a contractor, so I'm sure some of you can understand the beginning of the year budget juggling that goes on), it's back to the blog to finally get some real functionality into this program.

First, some administrative notes, I've updated the CSS for the blog template to help out with the coding. I've tested it in Firefox 2.0 and IE 7, and the font size issue should be fixed for all users. Please let me know if the code text is too small still. This will apply to Part 3 and everything from here on out.

I'm also working on a more advanced tutorial series that will create a pilotable ground vehicle using SimConnect and XML gauges. Look for that to start over the next month.

Right now, we have the ability to connect to FSX, and some error handling. Now we are going to get notification of an event, and respond to it. For this, we will need to cover some background information.

Remember how we talked before about how SimConnect programming is event-driven? Well, it goes a bit deeper than that. We can't really respond directly to FSX events, because we don't see them happening. Those events are created and handled on the FSX side of things. What we can do is use SimConnect to "map" a "SimEvent", something that happens in FSX, to a "client event" which is in our program.

So that's nice, and you can sort of tie those things together in a couple of ways. Right now, we just want to be notified when the SimEvent occurs. So we add our client event (mapped to the SimEvent) to what is called a Notification Group. Once we add that, we will get a RECV_EVENT system message (like our RECV_OPEN and RECV_QUIT messages from the last tutorial) that tells us that one of our events have fired off.

Once we get this message, we are able to handle the data properly. In this case, we are just going to toggle a message to indicate Landing Gear status. To do this, we are going to be looking at an event called "GEAR_TOGGLE", not surprisingly.

The first thing we want to do is set up the variables that we want to use. First is a bit of a placeholder. We are going to assume right now that we are starting the program with the plane on the runway with the gear down. In our next tutorial, we will be triggering the gear up and down ourselves, and be able to use some data requests to determine when the gear are fully extended or retracted.

For now, as I said, we will make some assumptions. First, that the program is starting with the gear down, and secondly, that you are going to be using the "Toggle Gear" control, which is G on your keyboard by default.

On to the programming:

I realized earlier that I wasn't doing a real spot-on job of commenting my code, which is a major no-no to me. If your code is ever for public consumption, even in a team programming environment, I say comment every line that needs it. Never make the assumption that the other person looking at or working on your code understands what you were thinking when you wrote it.

A comment is simply made by putting "//" at the beginning of the comment. Anything after the slashes will be ignored when you compile the code. For comments spanning multiple lines, use "/*" to begin and "*/" to end.

Some examples:

```
//This is a one line comment
int z = x + y;
```

The first line is a comment, and will be ignored, that is to say, the compiler won't try to think of it as code.

```
/*
This is a mult-line comment
```

```
T
*
```

Once again, same principle. The first line of code we will be adding is a "bool" variable. Instead of an integer having a value like "10" or "-250", or a string equaling "Sam" or whatever, a bool is simply a true/false variable. It only will equal true or false. We are going to use this variable to describe if the gear are up. If so, it will equal true. Since we are assuming that the program is starting with the gear down, it starts off as false.

```
SimConnect simconnect;
const int WM_USER_SIMCONNECT = 0x0402;

// Our current gear setting
bool gear_up = false;
```

This goes right at the beginning of our Form1 class. I included the WM_USER_SIMCONNECT variable to let you know where to put the new variable and comment.

Like I said before, we are going to map a server event "TOGGLE_GEAR" to our own client event. For the sake of simplicity, we'll call them the same thing. We're going to place our client event in what is called an "Enumeration". This is another type of variable that, simply put, is a list of values.

The classic example is selecting a character type for a computer RPG game, but I'll put it more into context. Say you are writing your SimConnect program and you have a section to select the plane you are going to fly. You can create an enumeration that contains the names of the available planes. It would look something like this:

```
enum AirplaneTypes
{
   PiperCub,
   Boeing738,
   Boeing747
}
```

What you are doing here is creating a new data type, called AirplaneTypes, that can have only one of the values listed inside the brackets. For example, you could later create a variable to track what the user has chosen from a drop down menu:

```
if ( DropdownMenu.Text == "Piper Cub" )
{
   AirplaneTypes SelectedPlane = AirplaneTypes.PiperCub;
}
```

Basically, what we are saying with this code is "If we've chosen Piper Cub from the dropdown menu we created, then pick the PiperCub value from our enumeration.

This is just to illustrate the basic principles. For the time being, we are only going to have one entry in each of our two enumerations. This code goes right after the last section.

```
// Events
enum EVENTS
{
    GEAR_TOGGLE
}

// Notification Group
enum NGROUP
{
    GEAR_GROUP
}
```

pretty much just names to reference things by.

In our button_Connect_Click function, we connect to SimConnect and set up some event handlers for OnRecvOpen and OnRecvQuit. There's one more handler we need, and that's "OnRecvEvent". The line of code we will add there looks like this:

```
// Receive events
simconnect.OnRecvEvent += new SimConnect.RecvEventEventHandler(simconnect_OnRecvEvent);
```

Pretty much the same as our other event handlers. If you tabbed through the autocomplete for that line, it will have created the simconnect_OnRecvEvent function. We'll leave that alone for right now. We'll be coming back to it in a minute after we get our event notification set up.

If we look down into the function to handle the Open event (simconnect_OnRecvOpen), you'll see that all we do is change the text on our textbox and connect button. Once we've established our connection, we'll want to use this program to set up our event notification. I'll toss out the code and then go over it:

```
// Set up EVENT handling for landing gear
simconnect.MapClientEventToSimEvent (EVENTS.GEAR_TOGGLE, "GEAR_TOGGLE");
simconnect.AddClientEventToNotificationGroup(NGROUP.GEAR_GROUP , EVENTS.GEAR_TOGGLE, false);
```

Ok, these are two completely new functions, so let's go into them step by step. Luckily, the SimConnect API is nice enough to use very human-readable functions. The first function obviously maps a client event to a Sim event. The parameters follow that same pattern. First we see EVENTS.GEAR_TOGGLE. If you recall what I said about enumerations, that means the EVENTS enum, and the value GEAR_TOGGLE. This is the identifier that we are sending SimConnect to tie in to the Sim Event. Following that is a string value representing the name of the Sim Event. These are all listed in the SDK documents. The layout is pretty easy to follow in there. First is the keypress event, next is the actual sim event name, and then an explanation of what it does. Once again, the names are pretty easy to understand. When you press the Gear Toggle key, it causes the "GEAR_TOGGLE" event to fire off.

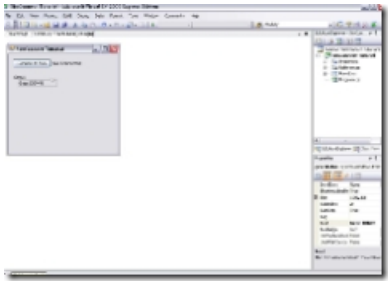
In our function, we are associating the "GEAR_TOGGLE" event to our local EVENTS.GEAR_TOGGLE identifier. Let's break down what that means:

- In FSX, you press the Gear Toggle key.
- FSX fires off the "GEAR_TOGGLE" event
- SimConnect is watching for that event, and sends a message to our program with the name of our local client event (EVENTS.GEAR_TOGGLE)
- Our program recieves the event with the event ID (EVENTS.GEAR_TOGGLE) and runs the OnRecvEvent function

I hope that clears everything up. I'm reading back and it all sounds like it could be a bit confusing. So, we've mapped our client event to the SimEvent, and now we tell SimConnect to watch for that event to fire. The next line (AddClientEventToNotificationGroup) does that. As you can see, the first parameter in the function is our notification group identifier (NGROUP.GEAR_GROUP) and then the name of our client event that we want to be notified about. There's a third parameter, "false", which asks whether or not the event should be masked. This means do we want to hide the event from other clients. It's actually possible to hide it from Flight Simulator itself, but we don't want to worry about that now.

Now our events and groups are set up and properly mapped. At this point, if we run the program and raise the landing gear in FSX, SimConnect will send through our OnRecvEvent message. But we aren't doing anything with it yet. Time to fix that.

label above the textbox called "Status", feel free to position or add whatever you want. Here's a screen shot of my design screen (click for full size):



Last step to finish up this tutorial, which has gotten far too long, is to implement our simconnect_OnRecvEvent function. Once again, I'll post the code and then explain it:

```
void simconnect_OnRecvEvent(SimConnect sender, SIMCONNECT_RECV_EVENT data)
{
    if (data.uEventID == (uint)EVENTS.GEAR_TOGGLE)
    {
        if (gear_up)
        {
            gearstatus.Text = "Gear: DOWN";
            gear_up = false;
        }
        else
        {
            gearstatus.Text = "Gear: UP";
            gear_up = true;
        }
    }
}
```

Looking at the first line inside the function, there's an if statement. Might look a bit confusing at first, but it's actually pretty easy to figure out. The "data" variable is the information that SimConnect is sending in the system message to our program. It contains a lot of information, depending on what sort of message we are receiving. In this case, what we are looking for is the ID of the event that fired off. That's stored in the variable "uEventID". You can see C#'s "dot syntax" in the works there; "data" is the name of the chunk of information, and uEventID is the specific variable we want. We are checking to see if it is equal to EVENTS.GEAR_TOGGLE. The "(uint)" section is what's called a "cast". Inside the enumeration EVENTS, each entry is given a number to identify it in the enumeration. By "casting" that value as a uint (or unsigned integer, basically just a number), we can match it up with the number of the event that SimConnect sends us. Casting just means that we are changing from one type of variable to another so that it matches up.

If it is the EVENTS.GEAR_TOGGLE event being sent to us, then we check back to that bool variable we made at the beginning. If it is true (a conditional statement like an if statement only executes the code inside if the part inside the parentheses is true...that can if 1+2=3 or simply a bool like gear_up, that will either be true of false), then the gear is currently up and is extending, so we set the text of our new textbox to indicate that the gear is down, and set gear_up to false.

Else (if gear_up is false) then we simply do the opposite. "Gear: UP" and true. As I said earlier, this is assuming that you are starting with it down. If you start mid-flight with your gear up, the value in the program will always

b

Save your work and give it a try. Whenever you toggle the gear in FSX, the status on your program should change to reflect that.

Next time, we are going to implement a few changes to this procedure. We will have a button on our program to flip the gear up or down, and will trigger the events ourself. We will also be reading the status of the gear using SimConnect's **RequestDataOnSimObject** function. This allows us to extract specific information from the sim. We'll even put in a green panel light into our program that will turn on while the gear is being toggled, and turn off when it it is done. It should be a big one, so look for it in the next few days.

As always, please leave your comments and questions here or email me at brian.gefrich@gmail.com

See you next time.

POSTED BY NOTASENATOR AT WEDNESDAY, JANUARY 24, 2007

2 COMMENTS:

Tim Gregson said...

Hey Brian,

Good to see you back and healthy again :->

Looks like a nice tutorial series you got going here so far, should help out lots of folks (possibly even me one of these days, if I ever get around to doing any SimConnect client stuff :->).

Looking forward to the next installment

Beatle

JANUARY 26, 2007 1:36:00 AM EST

William said...

Thank you for another great article. I can't wait for the more complex stuff to come as I'm new to C#. :)

FEBRUARY 1, 2007 12:29:00 PM EST

POST A COMMENT

<< Home