

```
1  //-----
2  //      Title:MASTER DEGREE THESIS by ANTONIO SCAZZI
3  //
4  //  Description:header file with the simulation main body
5  //-----
6  #pragma once
7  #include "globalvar.h"
8  #include "communicationmodule.h"
9
10 //funzione di callback per gestire gli eventi del simulatore
11 void CALLBACK MyDispatchProc(SIMCONNECT_RECV* pData, DWORD cbData, void* pContext)
12 {
13     switch (pData->dwID)
14     {
15         //code called on the connection with the simuation
16         case SIMCONNECT_RECV_ID_OPEN:
17         {
18             printf("Inizializzazione\n");
19             // Set up the data definition 1
20             hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Title", NULL,
21             SIMCONNECT_DATATYPE_STRING256);
22             hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Sim Time", "seconds");
23             hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Plane Altitude", "feet");
24             hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Plane Latitude", "degrees");
25             hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Plane Longitude", "degrees");
26             hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Plane Pitch Degrees", "degrees");
27             hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Plane Bank Degrees", "degrees");
28             hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Plane Heading Degrees True",
29             "degrees");
30             hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "GROUND VELOCITY", "Knots");
31             hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Velocity Body X", "feet per second");
32             hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Velocity Body Y", "feet per second");
33             hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Velocity Body Z", "feet per second");
```

```
32     hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Elevator Deflection", "degrees");
33     hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Rudder Deflection", "degrees");
34     hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Aileron Left Deflection", "degrees");
35     hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "General Eng Throttle Lever Position:1", "percent");
36     hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "TRANSPONDER CODE:1", "BC016");
37     hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Incidence alpha", "degrees");
38     hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Incidence beta", "degrees");
39     hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Rotation velocity body X", "radians per second");
40     hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Rotation velocity body Z", "radians per second");
41     hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Rotation velocity body Y", "radians per second");
42     hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Acceleration Body Z", "feet per second squared");
43     hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Acceleration Body X", "feet per second squared");
44     hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Acceleration Body Y", "feet per second squared");
45     hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "RELATIVE WIND VELOCITY BODY X", "feet per second squared");
46     hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "RELATIVE WIND VELOCITY BODY Y", "feet per second squared");
47     hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "RELATIVE WIND VELOCITY BODY Z", "feet per second squared");
48
49
50     // Set up the data definition 2
51     hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_2, "Plane Heading Degrees True", "degrees");
52
53     //link the event in the simulator to the event in the addon
```

```
54     hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_THROTTLE_SET, "THROTTLE_SET");
55     hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_ELEVATOR_SET, "ELEVATOR_SET");
56     hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_AILERON_SET, "AILERON_SET");
57     hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_RUDDER_SET, "RUDDER_SET");
58     hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_FLAPS_SET, "FLAPS_SET");
59     hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_PARKING_BRAKES_SET, "PARKING_BRAKES");
60     hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_LANDING_GEAR, "GEAR_UP");
61     hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_TRANSPONDER, "XPNDR_SET");
62
63
64     //INPUTS
65     // simulator keys
66     hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_2, "HOTAS_KEY_A0");
67     hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_2);
68
69     hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_3, "HOTAS_KEY_A1");
70     hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_3);
71
72     //keyboard keys
73     // Map event, add to notification group map the keys to the private events and activate them
74     hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_0);
75     hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_0);
76     hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_0, "VK_COMMA", EVENT_0);
77     hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_0, SIMCONNECT_STATE_ON);
78
79     hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_1);
80     hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_1);
81     hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_1, "q", EVENT_1);
82     hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_1, SIMCONNECT_STATE_ON);
83
84     hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_5);
85     hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_5);
86     hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_3, "w", EVENT_5);
```

```
87     hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_3, SIMCONNECT_STATE_ON);
88
89     hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_6);
90     hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_6);
91     hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_4, "a", EVENT_6);
92     hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_4, SIMCONNECT_STATE_ON);
93
94     hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_7);
95     hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_7);
96     hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_5, "s", EVENT_7);
97     hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_5, SIMCONNECT_STATE_ON);
98
99
100    // set the notification group priority
101    hr = SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP_0, SIMCONNECT_GROUP_PRIORITY_HIGHEST);
102
103    //take some inicial values
104    initial_transponder = UserPlane.transponderCode;
105    initial_simtime = UserPlane.simtime;
106    initial_latitude = UserPlane.latitude;
107    initial_longitude = UserPlane.longitude;
108
109
110    // Subscribe to Pause and Unpause events
111    SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_SIM_PAUSED, "Pause");
112
113
114    printf("Initializazione completata\n");
115
116    }
117    break;
118
119    //code to handle events received in a SIMCONNECT_RECV_EVENT structure.
```

```
120     case SIMCONNECT_RECV_ID_EVENT:
121     {
122         SIMCONNECT_RECV_EVENT* evt = (SIMCONNECT_RECV_EVENT*)pData;
123         switch (evt->uEventID)
124         {
125             case EVENT_SIM_PAUSED:
126             {
127                 if (evt->dwData == 1) {
128                     printf("Simulatore in Pausa\n");
129                     flag_isrunning = 0;
130                 }
131                 else {
132                     printf("Simulatore in Esecuzione\n");
133                     flag_isrunning = 1;
134                 }
135             }
136             break;
137
138             case EVENT_SIM_UNPAUSED:
139             {
140             }
141             break;
142
143             case EVENT_0:
144             {
145                 //event 0 è assegnato alla virgola
146                 if(flag_decollo==0)
147                 {
148                     flag_decollo = 1;
149                     printf("Autopilota di immissione in crociera attivo\n");
150                 }
151             }
152         }
```

```
153         }
154         else
155         {
156             printf("Comandi Manuali\n");
157             flag_decollo = 1001;
158         }
159     }
160     break;
161
162
163     case EVENT_1:
164     {
165         //event 1 è assegnato alla q
166         printf("q premuto\n");
167
168         if (flag_pitch == 0)
169         {
170
171             flag_pitch = 1;
172             pitchang = 1000;
173         }
174         else
175         {
176
177             flag_pitch = 0;
178             pitchang = 0;
179         }
180     }
181
182     }
183     break;
184
185     case EVENT_5:
```

```
186         {
187             //event 1 è assegnato alla q
188             printf("w premuto\n");
189
190
191         }
192     }
193     break;
194
195     case EVENT_6:
196     {
197         //event 6 assegnato alla a
198         printf("a premuto\n");
199
200         if (flag_roll == 0)
201         {
202
203             flag_roll = 1;
204             bankang = -20.0;
205         }
206         else
207         {
208
209             flag_roll = 0;
210             bankang = 0;
211
212         }
213     }
214     break;
215
216     case EVENT_7:
217     {
```

```
219         //event 1 è assegnato alla q
220         printf("s premuto\n");
221
222
223
224     }
225     break;
226
227
228     case EVENT_2:
229     {
230         flag_decollo = 1;
231         printf("Autopilota di immissione in crociera attivo\n");
232         // The message text to be displayed
233         const char* message = "AUTOPILOTA DI DECOLLO E IMMISSIONE IN CROCIERA ATTIVO";
234         hr=SimConnect_Text(hSimConnect, SIMCONNECT_TEXT_TYPE_MESSAGE_WINDOW,10.0,EVENT_PRINT_1, (DWORD)↵
            (strlen(message) + 1), (void*)message);
235         hr = SimConnect_ClearNotificationGroup(hSimConnect, GROUP_1);
236
237     }
238     break;
239
240     case EVENT_3:
241     {
242         flag_decollo = 1001;
243         printf("Passaggio a comandi manuali\n");
244         // The message text to be displayed
245         const char* message2 = "COMANDI MANUALI";
246         hr = SimConnect_Text(hSimConnect, SIMCONNECT_TEXT_TYPE_MESSAGE_WINDOW, 10.0, EVENT_PRINT_2, ↵
            (DWORD)(strlen(message2) + 1), (void*)message2);
247         hr = SimConnect_ClearNotificationGroup(hSimConnect, GROUP_2);
248
249     }
```

```
250         break;
251
252
253
254         default:
255         break;
256     }
257 }
258 break;
259
260 // if a call to SimConnect_RequestDataOnSimObjectType succeed this event opens.
261 case SIMCONNECT_RECV_ID_SIMOBJECT_DATA_BYTYPE:
262 {
263     SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE* pObjData = (SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE*)pData;
264     switch (pObjData->dwRequestID)
265     {
266         //richiesta di info su altri velivoli
267         case REQUEST_0:
268         {
269             DWORD Object_flag = 0;
270             for (int i = 1; i <= 2; i++)
271             {
272                 pObjData->dwenrynumber = i;
273                 DWORD ObjectID = pObjData->dwObjectID;
274
275                 if (pObjData->dwObjectID != UserID && pObjData->dwObjectID != Object_flag)
276                 {
277                     flag_dronefound = 1;
278                     Object_flag = ObjectID;
279                     ObjectID2 = ObjectID;
280                     V2 = (ObjectDataStruct*)&pObjData->dwData;
281                     if (SUCCEEDED(StringCbLengthA(&V2->title[0], sizeof(V2->title), NULL))) // security check
```

```
282     {
283         flag_dronefound = 1;
284         OtherPlane = *V2;
285
286         OtherPlane.velocity = sqrt(OtherPlane.velocityX * OtherPlane.velocityX +
OtherPlane.velocityY * OtherPlane.velocityY + OtherPlane.velocityZ *
OtherPlane.velocityZ);
287         //print a schermo delle caratteristiche
288         printf("DR Alt=%f Pitch=%f Bank=%f Heading=%f V=%.1f VX=%.1f VY=%.1f VZ=%.1f Elev=
%.2f Rudd=%.2f Ail=%.2f Thrott=%.1f Transp=%.1f alpha=%.5f beta=%.5f\n", V2->altitude,
V2->pitch, V2->bank, V2->heading, OtherPlane.velocity, V2->velocityX, V2->velocityY,
V2->velocityZ, V2->elevator, V2->rudder, V2->aileron, V2->throttle, V2-
>transponderCode, V2->alpha, V2->beta);
289     }
290 }
291 }
292 }
293 break;
294
295 //richiesta di info sul velivolo principale
296 case REQUEST_1:
297 {
298     UserID = pObjData->dwObjectID;
299     V1 = (ObjectDataStruct*)&pObjData->dwData;
300     if (SUCCEEDED(StringCbLengthA(&V1->title[0], sizeof(V1->title), NULL))) // security check
301     {
302         //passo le variabili ad un vettore utilizzabile
303         UserPlane = *V1;
304         UserPlane.velocity = sqrt(UserPlane.velocityX * UserPlane.velocityX + UserPlane.velocityY *
UserPlane.velocityY + UserPlane.velocityZ * UserPlane.velocityZ);
305
306         //print a schermo delle caratteristiche
307         printf("C2 Alt=%f Pitch=%f Bank=%f Heading=%f V=%.1f VX=%.1f VY=%.1f VZ=%.1f Elev=%.2f
```

```
Rudd=%.2f Ail=%.2f Thrott=%.1f Transp=%.1f alpha=%.5f beta=%.5f\n", V1->altitude, V1->pitch, V1->bank, V1->heading, UserPlane.velocity, V1->velocityX, V1->velocityY, V1->velocityZ, V1->elevator, V1->rudder, V1->aileron, V1->throttle, V1->transponderCode, V1->alpha, V1->beta);

308
309     }
310 }
311 break;
312 default:
313 break;
314 }
315 }
316 break;
317
318
319 //code to handle errors received in a SIMCONNECT_RECV_EXCEPTION structure.
320 case SIMCONNECT_RECV_ID_EXCEPTION:
321 {
322     SIMCONNECT_RECV_EXCEPTION* except = (SIMCONNECT_RECV_EXCEPTION*)pData;
323
324     switch (except->dwException)
325     {
326     case SIMCONNECT_EXCEPTION_ERROR:
327     {
328         printf("\nError");
329     }
330     break;
331     default: {}
332     break;
333     }
334 }
335 break;
336
```

```
337 //code to handle exiting the application
338 case SIMCONNECT_RECV_ID_QUIT:
339 {
340     flag_quit = 1;
341 }
342 break;
343
344 // code to handle the case where an unexpected message is received
345 default:
346 {
347     printf("\nReceived:%d", pData->dwID);
348 }
349 break;
350 }
351 }
352
```