

# Orbiter API

## 2010 Edition

Generated by Doxygen 1.5.6

Mon Aug 29 09:30:26 2011

## Contents

<b>1</b>	<b>Orbiter API Reference Manual</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Contents . . . . .	1
<b>2</b>	<b>Planet Modules</b>	<b>2</b>
2.1	First Steps: . . . . .	2
2.2	The CELBODY interface class . . . . .	3
<b>3</b>	<b>Graphics Client Development</b>	<b>4</b>
<b>4</b>	<b>Deleted and obsolete functions and methods</b>	<b>5</b>
<b>5</b>	<b>Basics of orbital mechanics</b>	<b>5</b>
5.1	Contents . . . . .	5
5.2	Elliptic orbits . . . . .	5
5.3	Elliptic orbits . . . . .	5
5.4	The orbit in space . . . . .	7
5.5	Mean longitude . . . . .	7
5.6	Kepler's equation . . . . .	8
5.7	True and eccentric anomaly . . . . .	8
5.8	Kepler's equation . . . . .	8
<b>6</b>	<b>Particle Streams HowTo</b>	<b>8</b>
6.1	Adding particle stream support . . . . .	9
6.2	Attaching and detaching streams . . . . .	9
6.3	Deleting streams . . . . .	9
<b>7</b>	<b>Vessel module concepts</b>	<b>9</b>
7.1	Docking port management . . . . .	9
7.2	Attachment management . . . . .	10
<b>8</b>	<b>Todo List</b>	<b>11</b>
<b>9</b>	<b>Deprecated List</b>	<b>11</b>
<b>10</b>	<b>Bug List</b>	<b>14</b>
<b>11</b>	<b>Module Index</b>	<b>14</b>
11.1	Modules . . . . .	14

<b>12 Class Index</b>	<b>17</b>
12.1 Class Hierarchy . . . . .	17
<b>13 Class Index</b>	<b>19</b>
13.1 Class List . . . . .	19
<b>14 File Index</b>	<b>21</b>
14.1 File List . . . . .	21
<b>15 Module Documentation</b>	<b>22</b>
15.1 Ephemeris data format bitflags . . . . .	22
15.1.1 Detailed Description . . . . .	22
15.2 Configuration parameter identifiers . . . . .	22
15.2.1 Detailed Description . . . . .	22
15.2.2 Define Documentation . . . . .	23
15.3 Render parameter identifiers . . . . .	26
15.3.1 Detailed Description . . . . .	26
15.3.2 Define Documentation . . . . .	26
15.4 Bit flags for planetarium mode elements . . . . .	26
15.5 Bit flags for blitting operations . . . . .	27
15.5.1 Define Documentation . . . . .	27
15.6 Some useful general constants . . . . .	28
15.7 Defines and Enumerations . . . . .	28
15.8 Structure definitions . . . . .	29
15.9 Handles . . . . .	30
15.10 Identifiers for special render surfaces . . . . .	31
15.10.1 Define Documentation . . . . .	31
15.11 Vectors and matrices . . . . .	31
15.11.1 Detailed Description . . . . .	31
15.11.2 Function Documentation . . . . .	33
15.12 Surface and texture attributes . . . . .	41
15.12.1 Detailed Description . . . . .	41
15.13 Mesh group editing flags . . . . .	41
15.13.1 Detailed Description . . . . .	41
15.14 Bitflags for EXHAUSTSPEC flags field. . . . .	43
15.14.1 Detailed Description . . . . .	43
15.15 Local lighting interface . . . . .	43
15.15.1 Detailed Description . . . . .	43

15.16	Light beacon shape parameters . . . . .	44
15.16.1	Detailed Description . . . . .	44
15.17	Listentryflag . . . . .	44
15.18	Listclbkflag . . . . .	44
15.19	Animation flags . . . . .	44
15.19.1	Detailed Description . . . . .	44
15.20	Identifiers for frames of reference . . . . .	45
15.20.1	Enumeration Type Documentation . . . . .	45
15.21	Thruster and thruster-group parameters . . . . .	45
15.21.1	Enumeration Type Documentation . . . . .	45
15.22	Airfoil orientation . . . . .	46
15.22.1	Enumeration Type Documentation . . . . .	46
15.23	Aerodynamic control surface types . . . . .	47
15.23.1	Enumeration Type Documentation . . . . .	47
15.24	Control surface axis orientation . . . . .	47
15.24.1	Define Documentation . . . . .	48
15.25	Identifiers for visual events . . . . .	48
15.25.1	Detailed Description . . . . .	48
15.26	Navigation mode identifiers . . . . .	49
15.26.1	Detailed Description . . . . .	49
15.27	Manual control mode identifiers . . . . .	49
15.27.1	Detailed Description . . . . .	49
15.28	Manual control device identifiers . . . . .	50
15.28.1	Detailed Description . . . . .	50
15.29	RCS mode identifiers . . . . .	50
15.29.1	Detailed Description . . . . .	50
15.30	HUD mode identifiers . . . . .	51
15.30.1	Detailed Description . . . . .	51
15.31	MFD mode identifiers . . . . .	51
15.31.1	Detailed Description . . . . .	51
15.32	MFD identifiers . . . . .	52
15.33	Panel neighbour identifiers . . . . .	53
15.33.1	Detailed Description . . . . .	53
15.34	Panel redraw event identifiers . . . . .	54
15.34.1	Detailed Description . . . . .	54
15.35	Mouse event identifiers . . . . .	54

15.35.1 Detailed Description . . . . .	54
15.36 Panel area texture mapping identifiers . . . . .	55
15.36.1 Detailed Description . . . . .	55
15.37 Generic vessel message identifiers . . . . .	56
15.38 Vessel mesh visibility flags . . . . .	56
15.38.1 Detailed Description . . . . .	56
15.39 Navigation radio transmitter types . . . . .	57
15.39.1 Detailed Description . . . . .	57
15.40 Object parameter flags . . . . .	57
15.40.1 Detailed Description . . . . .	57
15.41 Orbiter API interface methods . . . . .	58
15.41.1 Detailed Description . . . . .	58
15.41.2 Function Documentation . . . . .	60
15.42 Object access functions . . . . .	65
15.42.1 Function Documentation . . . . .	66
15.43 Vessel creation and destruction . . . . .	73
15.43.1 Function Documentation . . . . .	73
15.44 Body functions . . . . .	75
15.44.1 Function Documentation . . . . .	75
15.45 Vessel functions . . . . .	77
15.45.1 Function Documentation . . . . .	79
15.46 Coordinate transformations . . . . .	92
15.46.1 Function Documentation . . . . .	93
15.47 Camera functions . . . . .	96
15.47.1 Function Documentation . . . . .	97
15.48 Functions for planetary bodies . . . . .	102
15.48.1 Detailed Description . . . . .	102
15.48.2 Function Documentation . . . . .	103
15.49 Surface base interface . . . . .	109
15.49.1 Function Documentation . . . . .	109
15.50 Time functions . . . . .	112
15.50.1 Function Documentation . . . . .	112
15.51 Navigation radio transmitter functions . . . . .	116
15.51.1 Function Documentation . . . . .	117
15.52 Script interpreter functions . . . . .	120
15.52.1 Function Documentation . . . . .	121

15.53 Visual and mesh functions . . . . .	122
15.53.1 Typedef Documentation . . . . .	124
15.53.2 Function Documentation . . . . .	124
15.54 HUD, MFD and panel functions . . . . .	133
15.54.1 Function Documentation . . . . .	135
15.55 Drawing support functions . . . . .	146
15.55.1 Function Documentation . . . . .	147
15.56 Surface functions . . . . .	151
15.56.1 Function Documentation . . . . .	152
15.57 Custom MFD mode definition . . . . .	156
15.57.1 Function Documentation . . . . .	157
15.58 Virtual cockpit functions . . . . .	159
15.58.1 Function Documentation . . . . .	159
15.59 Customisation - custom menu, dialogs . . . . .	163
15.59.1 Function Documentation . . . . .	164
15.60 File IO Functions . . . . .	170
15.60.1 Function Documentation . . . . .	171
15.61 Utility functions . . . . .	179
15.61.1 Function Documentation . . . . .	179
15.62 User input functions . . . . .	180
15.62.1 Function Documentation . . . . .	180
15.63 Onscreen annotations . . . . .	180
15.63.1 Detailed Description . . . . .	180
15.63.2 Function Documentation . . . . .	181
15.64 Obsolete functions . . . . .	183
15.64.1 Function Documentation . . . . .	184
15.65 Keyboard key identifiers . . . . .	186
15.66 Logical key ids . . . . .	192
15.67 Top-level module callback functions . . . . .	199
15.67.1 Detailed Description . . . . .	199
15.68 General module callback functions . . . . .	199
15.68.1 Detailed Description . . . . .	199
15.68.2 Function Documentation . . . . .	199
15.69 Vessel module callback functions . . . . .	200
15.69.1 Detailed Description . . . . .	200
15.69.2 Function Documentation . . . . .	201

15.70Plugin module callback functions . . . . .	201
15.70.1 Detailed Description . . . . .	201
15.70.2 Function Documentation . . . . .	202
<b>16 Class Documentation</b>	<b>206</b>
16.1 ANIMATION Struct Reference . . . . .	206
16.1.1 Detailed Description . . . . .	206
16.2 ANIMATIONCOMP Struct Reference . . . . .	207
16.2.1 Detailed Description . . . . .	207
16.3 ATMCONST Struct Reference . . . . .	207
16.3.1 Detailed Description . . . . .	208
16.4 ATMOSPHERE Class Reference . . . . .	209
16.4.1 Detailed Description . . . . .	209
16.4.2 Member Enumeration Documentation . . . . .	210
16.4.3 Constructor & Destructor Documentation . . . . .	210
16.4.4 Member Function Documentation . . . . .	210
16.5 ATMOSPHERE::PRM_IN Struct Reference . . . . .	211
16.5.1 Detailed Description . . . . .	212
16.6 ATMOSPHERE::PRM_OUT Struct Reference . . . . .	212
16.6.1 Detailed Description . . . . .	212
16.7 ATMPARAM Struct Reference . . . . .	213
16.7.1 Detailed Description . . . . .	213
16.8 BEACONLIGHTSPEC Struct Reference . . . . .	213
16.8.1 Detailed Description . . . . .	214
16.9 oapi::Brush Class Reference . . . . .	215
16.9.1 Detailed Description . . . . .	215
16.9.2 Constructor & Destructor Documentation . . . . .	215
16.10CELBODY Class Reference . . . . .	216
16.10.1 Detailed Description . . . . .	216
16.10.2 Member Function Documentation . . . . .	217
16.11CELBODY2 Class Reference . . . . .	220
16.11.1 Detailed Description . . . . .	220
16.11.2 Constructor & Destructor Documentation . . . . .	222
16.11.3 Member Function Documentation . . . . .	222
16.12COLOUR4 Struct Reference . . . . .	225
16.12.1 Detailed Description . . . . .	225
16.13oapi::DrawingTool Class Reference . . . . .	226

16.13.1 Detailed Description . . . . .	226
16.14 ELEMENTS Struct Reference . . . . .	226
16.14.1 Detailed Description . . . . .	226
16.15 ENGINESTATUS Struct Reference . . . . .	227
16.15.1 Detailed Description . . . . .	227
16.16 EXHAUSTSPEC Struct Reference . . . . .	228
16.16.1 Detailed Description . . . . .	228
16.17 ExternMFD Class Reference . . . . .	229
16.17.1 Detailed Description . . . . .	229
16.17.2 Constructor & Destructor Documentation . . . . .	231
16.17.3 Member Function Documentation . . . . .	231
16.18 FogParam Struct Reference . . . . .	233
16.18.1 Detailed Description . . . . .	234
16.19 oapi::Font Class Reference . . . . .	234
16.19.1 Detailed Description . . . . .	234
16.19.2 Member Enumeration Documentation . . . . .	235
16.19.3 Constructor & Destructor Documentation . . . . .	235
16.19.4 Member Function Documentation . . . . .	236
16.20 oapi::GraphicsClient Class Reference . . . . .	236
16.20.1 Detailed Description . . . . .	237
16.20.2 Constructor & Destructor Documentation . . . . .	243
16.20.3 Member Function Documentation . . . . .	244
16.21 oapi::GraphicsClient::LABELLIST Struct Reference . . . . .	275
16.21.1 Detailed Description . . . . .	275
16.22 oapi::GraphicsClient::VIDEODATA Struct Reference . . . . .	275
16.22.1 Detailed Description . . . . .	276
16.23 GraphMFD Class Reference . . . . .	276
16.23.1 Detailed Description . . . . .	277
16.23.2 Constructor & Destructor Documentation . . . . .	278
16.23.3 Member Function Documentation . . . . .	278
16.24 GROUPEDITSPEC Struct Reference . . . . .	280
16.24.1 Detailed Description . . . . .	281
16.25 HELPCONTEXT Struct Reference . . . . .	281
16.25.1 Detailed Description . . . . .	281
16.26 HUDPARAM Union Reference . . . . .	282
16.26.1 Detailed Description . . . . .	282

16.27oapi::IVECTOR2 Union Reference . . . . .	282
16.27.1 Detailed Description . . . . .	282
16.28LaunchpadItem Class Reference . . . . .	283
16.28.1 Detailed Description . . . . .	283
16.28.2 Member Function Documentation . . . . .	284
16.29LightEmitter Class Reference . . . . .	286
16.29.1 Detailed Description . . . . .	286
16.29.2 Constructor & Destructor Documentation . . . . .	288
16.29.3 Member Function Documentation . . . . .	288
16.30LISTENTRY Struct Reference . . . . .	292
16.30.1 Detailed Description . . . . .	292
16.31MATERIAL Struct Reference . . . . .	292
16.31.1 Detailed Description . . . . .	293
16.32MATRIX3 Union Reference . . . . .	293
16.32.1 Detailed Description . . . . .	293
16.33MESHGROUP Struct Reference . . . . .	294
16.33.1 Detailed Description . . . . .	294
16.34MESHGROUP_TRANSFORM Struct Reference . . . . .	295
16.34.1 Detailed Description . . . . .	295
16.35MESHGROUPEX Struct Reference . . . . .	296
16.35.1 Detailed Description . . . . .	296
16.36MFD Class Reference . . . . .	297
16.36.1 Detailed Description . . . . .	298
16.36.2 Constructor & Destructor Documentation . . . . .	299
16.36.3 Member Function Documentation . . . . .	300
16.37MFD2 Class Reference . . . . .	305
16.37.1 Detailed Description . . . . .	305
16.37.2 Constructor & Destructor Documentation . . . . .	306
16.37.3 Member Function Documentation . . . . .	307
16.38oapi::Module Class Reference . . . . .	310
16.38.1 Detailed Description . . . . .	310
16.38.2 Member Enumeration Documentation . . . . .	311
16.38.3 Constructor & Destructor Documentation . . . . .	311
16.38.4 Member Function Documentation . . . . .	312
16.39oapi::ModuleNV Class Reference . . . . .	315
16.39.1 Detailed Description . . . . .	316

16.39.2 Constructor & Destructor Documentation . . . . .	316
16.39.3 Member Function Documentation . . . . .	316
16.40NAVDATA Struct Reference . . . . .	318
16.40.1 Detailed Description . . . . .	318
16.41NTVERTEX Struct Reference . . . . .	319
16.41.1 Detailed Description . . . . .	319
16.42ORBITPARAM Struct Reference . . . . .	320
16.42.1 Detailed Description . . . . .	320
16.43oapi::ParticleStream Class Reference . . . . .	321
16.43.1 Detailed Description . . . . .	322
16.43.2 Constructor & Destructor Documentation . . . . .	323
16.43.3 Member Function Documentation . . . . .	323
16.44PARTICLESTREAMSPEC Struct Reference . . . . .	326
16.44.1 Detailed Description . . . . .	326
16.44.2 Member Enumeration Documentation . . . . .	328
16.44.3 Member Data Documentation . . . . .	328
16.45oapi::Pen Class Reference . . . . .	328
16.45.1 Detailed Description . . . . .	329
16.45.2 Constructor & Destructor Documentation . . . . .	329
16.46PointLight Class Reference . . . . .	329
16.46.1 Detailed Description . . . . .	330
16.46.2 Constructor & Destructor Documentation . . . . .	331
16.46.3 Member Function Documentation . . . . .	331
16.47oapi::ScreenAnnotation Class Reference . . . . .	332
16.47.1 Detailed Description . . . . .	333
16.47.2 Constructor & Destructor Documentation . . . . .	334
16.47.3 Member Function Documentation . . . . .	334
16.48oapi::Sketchpad Class Reference . . . . .	335
16.48.1 Detailed Description . . . . .	335
16.48.2 Member Enumeration Documentation . . . . .	337
16.48.3 Constructor & Destructor Documentation . . . . .	338
16.48.4 Member Function Documentation . . . . .	338
16.49SpotLight Class Reference . . . . .	347
16.49.1 Detailed Description . . . . .	348
16.49.2 Constructor & Destructor Documentation . . . . .	349
16.49.3 Member Function Documentation . . . . .	350

16.50VECTOR3 Union Reference . . . . .	350
16.50.1 Detailed Description . . . . .	350
16.51VESSEL Class Reference . . . . .	351
16.51.1 Detailed Description . . . . .	351
16.51.2 Constructor & Destructor Documentation . . . . .	375
16.51.3 Member Function Documentation . . . . .	376
16.52VESSEL2 Class Reference . . . . .	503
16.52.1 Detailed Description . . . . .	504
16.52.2 Constructor & Destructor Documentation . . . . .	506
16.52.3 Member Function Documentation . . . . .	506
16.53VESSEL3 Class Reference . . . . .	520
16.53.1 Detailed Description . . . . .	521
16.53.2 Constructor & Destructor Documentation . . . . .	522
16.53.3 Member Function Documentation . . . . .	522
16.54VESSEL4 Class Reference . . . . .	530
16.54.1 Detailed Description . . . . .	530
16.54.2 Constructor & Destructor Documentation . . . . .	530
16.54.3 Member Function Documentation . . . . .	530
16.55VESSELSTATUS Struct Reference . . . . .	531
16.55.1 Detailed Description . . . . .	531
16.55.2 Member Data Documentation . . . . .	532
16.56VESSELSTATUS2 Struct Reference . . . . .	533
16.56.1 Detailed Description . . . . .	534
16.56.2 Member Data Documentation . . . . .	535
16.57VESSELSTATUS2::DOCKINFOSPEC Struct Reference . . . . .	537
16.57.1 Detailed Description . . . . .	537
16.58VESSELSTATUS2::FUELSPEC Struct Reference . . . . .	537
16.58.1 Detailed Description . . . . .	538
16.59VESSELSTATUS2::THRUSTSPEC Struct Reference . . . . .	538
16.59.1 Detailed Description . . . . .	538
<b>17 File Documentation</b>	<b>538</b>
17.1 Orbitersdk/include/CelBodyAPI.h File Reference . . . . .	538
17.1.1 Detailed Description . . . . .	538
17.2 Orbitersdk/include/DrawAPI.h File Reference . . . . .	539
17.2.1 Detailed Description . . . . .	539
17.3 Orbitersdk/include/MFDAPI.h File Reference . . . . .	540

17.3.1 Detailed Description . . . . .	540
17.4 Orbitersdk/include/OrbiterAPI.h File Reference . . . . .	541
17.4.1 Detailed Description . . . . .	541
17.4.2 Function Documentation . . . . .	591
17.5 Orbitersdk/include/VesselAPI.h File Reference . . . . .	592
17.5.1 Detailed Description . . . . .	592
<b>18 Example Documentation</b>	<b>592</b>
18.1 clbkLoadStateEx.cpp . . . . .	592
18.2 clbkPreStep.cpp . . . . .	593
18.3 clbkSetStateEx.cpp . . . . .	593
<b>19 VESSEL2.cpp</b>	<b>594</b>

# 1 Orbiter API Reference Manual

Orbiter Software and documentation Copyright (C) 2010 Martin Schweiger

The Orbiter SDK Package contains this document in compiled HTML format (CHM) and in hyperlinked PDF format.

## 1.1 Introduction

This reference document contains the specification for the Orbiter Application Programming Interface (OAPI). The intended audience are developers who want to write DLL plugin modules for Orbiter. The document is *not* required for using Orbiter.

The programming interface allows the development of third party modules to enhance the functionality of the Orbiter core. Examples for modules are:

- new spacecraft (including custom flight models, instrument panels, etc.)
- new celestial bodies (including trajectory and atmospheric code)
- additional instruments, such as **MFD** (multifunctional display) modes
- global modules (networking, sound, etc.)

## 1.2 Contents

The following sections of this document are important for developers of Orbiter addon modules:

- Section **Orbiter API interface methods** contains a set of functions for getting and setting general simulation parameters in a running Orbiter simulation session. They are used by all types of plugin modules.
- The **VESSEL**, **VESSEL2** and **VESSEL3** classes are the base classes for vessel definitions. They are particularly useful for developers who want to create their own spacecraft plugins.

- The [MFD](#) and [MFD2](#) classes provide an interface for creating new multifunctional display modes.
- The [oapi::GraphicsClient](#) class provides an interface between Orbiter and external render engine modules. It is interesting for developers who want to substitute Orbiter's default graphics module with their own render engine plugin.

## 2 Planet Modules

Planet modules can be used to control the motion of a planet (or any other celestial body, such as a moon, the sun, or an asteroid) within the solar system. This allows to implement sophisticated analytic ephemerides solutions which take into account perturbations from other celestial objects.

Planets which are not controlled via a DLL module are updated directly by Orbiter. Depending on the settings in the definition file, Orbiter either uses an unperturbed 2-body approximation, resulting in a conic section trajectory (e.g. an ellipse), or uses a dynamic update procedure based on the gravitational forces acting on the planet. Both methods have limitations: the 2-body approach ignores perturbations and is only valid if no massive bodies other than the orbit reference object are nearby. The dynamic update accumulates numerical errors over time, causing the orbits slowly to diverge from the correct trajectories.

By using a planet module, analytic perturbation solutions can be used which avoid the shortcomings of the methods described above. Perturbation solutions typically describe the perturbed orbit of a planet by expressing the state vectors as a trigonometric series. These series are valid over a limited period of time, after which they start to diverge. Examples of perturbation solutions used in Orbiter are the VSOP87 solution for the 8 major planets and the sun, or the ELP2000 solution for the moon.

Planet modules have one additional function: They can be used to define some atmospheric parameters, such as temperature, pressure and density as a function of altitude. Additional functions may be added to the planet module interface in the future.

### 2.1 First Steps:

To start on your planet module, you should create a new "dynamic link library" project with your C++ compiler. Add the *Orbiter.lib* and *Orbitersdk.lib* files to the project (*found in Orbitersdk\lib*). Add *Orbitersdk\include* to your include path. Create a C++ source file for your project, and add the essential API interface functions:

```
#define ORBITER_MODULE
#include "OrbiterAPI.h"
#include "CelbodyAPI.h"

DLLCLBK void InitModule (HINSTANCE hModule)
{
    // module initialisation
}

DLLCLBK void ExitModule (HINSTANCE hModule)
{
    // module cleanup
}

DLLCKBK CELBODY *InitInstance (OBJHANDLE hBody)
{
    // instance initialisation
    return new MyPlanet;
}

DLLCLBK void ExitInstance (CELBODY *body)
```

```
{
    // instance cleanup
    delete (MyPlanet*)body;
}
```

The first line defining ORBITER\_MODULE is required to ensure that all initialisation functions are properly called by Orbiter.

[OrbiterAPI.h](#) contains the general API interface, and [CelBodyAPI.h](#) contains the planet module- specific interface, in particular the [CELBODY](#) class, which will be discussed below.

The [InitModule\(\)](#) and [ExitModule\(\)](#) methods are called only once per Orbiter session, when the DLL module is loaded or unloaded, respectively. They can be used to set up global parameters. You can omit them if your module doesn't need any such initialisation.

The [GetInstance\(\)](#) and [ExitInstance\(\)](#) functions are more important: You use them to create and destroy an instance of your planet class. This class is derived from [CELBODY](#). In this example, we called it MyPlanet.

## 2.2 The CELBODY interface class

All communication between Orbiter and your planet module will be conducted via the methods of the derived planet class. You overload the various callback functions of the [CELBODY](#) class to add the required functionality. Check the API Reference manual for a complete list of class methods. A typical implementation might look like this:

```
class MyPlanet: public CELBODY
{
public:
    MyPlanet();
    bool bEphemeris() const;
    void clbkInit (FILEHANDLE cfg);
    int clbkEphemeris (double mjd, int req, double *ret);
    int clbkFastEphemeris (double simt, int req, double *ret);
};

MyPlanet::MyPlanet () : CELBODY()
{
    // add constructor code here
}

bool MyPlanet::bEphemeris() const
{
    return true; // class supports ephemeris calculation
}

void MyPlanet::clbkInit (FILEHANDLE cfg)
{
    // read parameters from config file (e.g. tolerance limits, etc)
    // perform any required initialisation (e.g. read perturbation terms from data files)
}

int MyPlanet::clbkEphemeris (double mjd, int req, double *ret)
{
    // return planet position and velocity for Modified Julian date mjd in ret
}

int MyPlanet::clbkFastEphemeris (double simt, int req, double *ret)
{
    // return interpolated planet position and velocity for simulation time simt in ret
}
```

*clbkEphemeris()* and *clbkFastEphemeris()* are the functions which will contain the actual ephemeris calculations for the planet at the requested time. *clbkEphemeris()* is only called by Orbiter if the planet's state at an arbitrary time is required (for example by an instrument calculating the position at some future time). When Orbiter updates the planet's position for the next simulation time frame, the *clbkFastEphemeris()* function will be called instead. This means that *clbkFastEphemeris()* will be called at each frame, each time advancing the time by a small amount. This can be used for a more efficient calculation. Instead of performing a full series evaluation, which can be lengthy, you may implement an interpolation scheme which performs the full calculation only occasionally, and interpolates between these samples to return the state at an intermediate time.

For both functions, the requested type of data is specified as a group of EPHEM\_XXX bitflags in the req parameter. (see [CELBODY](#)) This can be any combination of position and velocity data for the celestial body itself and/or the barycentre of the system defined by the body and all its children (moons). The functions should calculate all required data, either in cartesian or polar coordinates, and fill the ret array with the results. ret contains 12 entries, used as follows:

ret[0-2]: true position  
ret[3-5]: true velocity  
ret[6-8]: barycentric position  
ret[9-11]: barycentric velocity

Only the fields requested by req need to be filled. In cartesian coordinates, the position fields must contain the x, y and z coordinates in [m], and the velocity fields must contain the velocities dx/dt, dy/dt, dz/dt in [m/s]. In spherical polar coordinates, the position fields must contain longitude j [rad], latitude q [rad] and radial distance r [AU], and the velocity fields must contain the polar velocities dj/dt [rad/s], dq/dt [rad/s] and dr/dt [AU/s].

The functions should indicate the fields actually calculated via the return value. This is in particular important if not all requests could be satisfied (e.g. position and velocity was requested, but only position could be calculated). The return value is interpreted as a bitflag that can contain the same EPHEM\_XXX flags as the req parameter. If all requests could be satisfied, it should be identical to req. In addition, the return value should contain additional flags indicating the properties of the returned data, including EPHEM\_POLAR if the data are returned as spherical polar coordinates, or EPHEM\_TRUEISBARY if the true and barycentric coordinates are identical (i.e. the celestial body does not have child bodies).

**Note:**

The older standalone module callback functions (opcXXX) are obsolete and should no longer be used.

**See also:**

[CELBODY](#)

## 3 Graphics Client Development

This page contains information for developers of plug-in graphics clients for the non-graphics version of Orbiter (Orbiter\_NG).

Graphics clients are DLL modules which contain the implementation of a client class derived from [oapi::GraphicsClient](#). They handle the device- specific aspects of rendering a 3-D window into the "orbiter world".

Contents:

- [Particle Streams HowTo](#)

## 4 Deleted and obsolete functions and methods

The following API function and class methods are no longer supported:

- [oapiGetStationByName\(\)](#)
- [oapiGetStationByIndex\(\)](#)
- [oapiGetStationCount\(\)](#)

## 5 Basics of orbital mechanics

This section of the manual contains a very brief summary of basic celestial mechanics. It is intended to clarify some of the concepts of various API functions in this reference document, but may also provide some useful general information for beginners.

### 5.1 Contents

[Elliptic orbits](#)

[The orbit in space](#)

[Kepler's equation](#)

### 5.2 Elliptic orbits

This page provides a summary of parameters for ideal 2-body orbital elements.

**Conic section:** The trajectory of an object under the influence of the gravitational field generated by a point mass follows a conic section. This may be either periodic (closed circular or elliptic orbit) or nonperiodic (open parabolic or hyperbolic orbit). The equation of a conic section with the focus in the origin is given in polar coordinates by

$$r = \frac{p}{1 + e \cos(\nu)}$$

with *eccentricity*  $e$  and *semi-latus rectum*  $p$ .

**Standard gravitational parameter:** In the following, the standard gravitational parameter is defined as the product of the gravitational constant  $G$  and the mass  $M$  of the central body at focus  $F$ :

$$\mu = GM$$

### 5.3 Elliptic orbits

Elliptic (closed) orbits are characterised by an eccentricity  $e < 1$ . A special case are circular orbits ( $e = 0$ ).

**Semi-major axis (a):** The longest semi-diameter of the ellipse. The distance from the centre (C) through one of the foci (F) to the edge of the ellipse (A). The semi-major axis can be calculated from the parameters of the conic section as

$$a = \frac{p}{1 - e^2}$$

**Semi-minor axis (b):** The shortest semi-diameter of the ellipse. The distance from the centre (C) to the edge of the ellipse, at right angles to the major axis. The semi-minor axis can be calculated from the parameters of the conic section as

$$b = \frac{p}{\sqrt{1 - e^2}} = a\sqrt{1 - e^2}$$

**Periapsis:** The periapsis (A) (perigee for Earth orbits, perilune for lunar orbits) is the lowest point of the orbit, i.e. the point of the ellipse closest to focus F. The periapsis distance  $r_{pe} = FA$  is given by

$$r_{pe} = \frac{p}{1 + e} = (1 - e)a$$

**Apoapsis:** The apoapsis (B) (apogee for Earth orbits, apolune for lunar orbits) is the highest point of the orbit, i.e. the point of the ellipse farthest from focus F. The apoapsis distance  $r_{ap} = FB$  is given by

$$r_{ap} = \frac{p}{1 - e} = (1 + e)a$$

**True anomaly:** The true anomaly ( $v$ ) of an orbiting object (P) is the angle AFP between P and apoapsis A, measured at F.

**Orbital period:** According to Kepler's third law, the square of the period T of an orbiting body is proportional to the cube of the semi-major axis a of the orbit. T is given by

$$T = 2\pi\sqrt{\frac{a^3}{\mu}}$$

**Orbital speed:** The orbital speed v as a function of radius r is given by

$$v = \sqrt{\mu \left( \frac{2}{r} - \frac{1}{a} \right)}$$

Maximum and minimum speed occur at periapsis and apoapsis, respectively:

$$v_{pe} = \sqrt{\frac{(1 + e)\mu}{(1 - e)a}}, \quad v_{ap} = \sqrt{\frac{(1 - e)\mu}{(1 + e)a}}$$

The mean orbital speed is given by

$$\bar{v} = \frac{2\pi a}{T} = \sqrt{\frac{\mu}{a}} = na$$

where the mean angular motion  $n$  is defined as

$$n = \frac{2\pi}{T}$$

**Specific orbital energy:** (or vis-viva energy) E is the sum of potential energy  $E_p$  and kinetic energy  $E_k$  of an orbiting body. E is constant along the orbit:

$$E = E_k + E_p = \frac{v^2}{2} - \frac{\mu}{r} = -\frac{1}{2} \frac{\mu^2}{h^2} (1 - e^2)$$

where h is the specific angular momentum of the orbiting body.

For specific types of orbit, E is given by

$$E = \begin{cases} -\frac{\mu}{2a} & \text{if } e < 1 \\ 0 & \text{if } e = 1 \\ \frac{\mu}{2a} & \text{if } e > 1 \end{cases}$$

## 5.4 The orbit in space

The orientation of the orbital trajectory in space, relative to the reference body, is defined by three parameters (in addition to the two parameters describing the shape):

- inclination
- longitude of ascending node
- longitude of periapsis

The position of the orbiting object along the orbit is defined by an additional parameter, the true longitude.

The orientation of an orbit in space is defined with respect to a frame of reference. For planetary orbits, the reference is usually given by the plane of the ecliptic and direction of the vernal equinox. For satellites in Earth orbit, the equatorial plane usually defines the reference plane.

**Inclination:** The *inclination*  $i$  defines the tilt of the orbital plane against the reference plane. The intersection of the orbital plane with the reference plane is denoted as the *line of nodes*.

**Ascending and descending node:** The line of nodes always passes through the orbit reference body (S). The *nodes*  $N_1$  and  $N_2$  are the points where the orbital trajectory intersects the reference plane. If the direction of orbit is such that the orbiting body passes the plane of the ecliptic from south to north at  $N_1$ , then  $N_1$  is the *ascending node*, and  $N_2$  is the *descending node*.

**Longitude of ascending node:** The angle between the reference direction  $\Upsilon$  and node  $N_1$  is the *longitude of the ascending node* ( $\theta$ ).

**Argument of periapsis:** The angle between node  $N_1$  and periapsis A is the *argument of periapsis* ( $\omega$ ).

**Longitude of periapsis:** The sum  $\varpi = \theta + \omega$  is called the *longitude of periapsis*.

**True longitude:** The sum of longitude of periapsis and true anomaly,

$$L = \varpi + \nu = \theta + \omega + \nu$$

is called the *true longitude* of the orbiting body.

## 5.5 Mean longitude

Consider a vector originating in S and moving in the plane of the orbit with mean angular velocity  $n$ , passing through point A at time  $t_0$ .

**Mean anomaly:** At time  $t$ , the vector is located at *mean anomaly*  $M = n(t - t_0)$  relative to periapsis A.

**Mean longitude:** The *mean longitude* of the orbiting body is the sum of mean anomaly and longitude of periapsis:

$$l = \theta + \omega + n(t - t_0) = \varpi + n(t - t_0)$$

The *mean longitude at the epoch* is defined as the mean longitude at  $t=0$ , given by

$$\varepsilon = \varpi - nt_0$$

The mean longitude can then be written as

$$l = nt + \varepsilon.$$

The mean anomaly is given by

$$M = n(t - t_0) = l - \varpi = nt + \varepsilon - \varpi$$

## 5.6 Kepler's equation

### 5.7 True and eccentric anomaly

To find the position P of an orbiting body at some time t, we need to find its true anomaly  $\nu$  at that time. Calculating true anomaly is not trivial for eccentric orbits, because the velocity of the orbiting body is continually changing.

**Eccentric anomaly:** Define a circle with radius  $a$  (semi-major axis of the orbit ellipse) whose centre coincides with the centre of the ellipse. Project object position P perpendicular to the semi-major axis onto the circle (Q). Then the *eccentric anomaly* E is defined as the angle ACQ between periapsis A and Q, measured at the centre C of the circle.

The relationship between orbit radius r and eccentric anomaly E is given by

$$r = a(1 - e \cos E)$$

The relationship between true anomaly  $\nu$  and E is given by

$$\tan \frac{\nu}{2} = \sqrt{\frac{1+e}{1-e}} \tan \frac{E}{2}$$

With these equations, position P can be calculated when eccentric anomaly E is known. E is calculated for a given time t by solving *Kepler's equation*.

## 5.8 Kepler's equation

Consider a vector rotating around C at constant angular velocity n, given by the orbiter's mean motion. If the vector passes A at time  $t_0$ , then its angle with A at time t is given by

$$M(t) = n(t - t_0)$$

M is called the *mean anomaly*. Kepler's equation defines a relation between mean anomaly M and eccentric anomaly E:

$$E(t) - e \sin E(t) = M(t) = n(t - t_0)$$

It cannot be solved for E in closed form, and must generally be solved iteratively.

## 6 Particle Streams HowTo

Particle streams are a component of Orbiter graphics clients (see [Graphics Client Development](#)).

Particle streams can be used to create visual effects for contrails, exhaust and plasma streams, reentry heating, condensation, etc.

The management of particle streams is almost entirely the responsibility of the graphics client. The orbiter core notifies the client only

- to request a new particle stream for a vessel object
- to detach a stream from its object (e.g. if the object is deleted)

The implementation details for the particle streams, including render options, are left to the client.

## 6.1 Adding particle stream support

To add particle stream support to a graphics client, the following steps are required:

- Create one or more classes derived from [oapi::ParticleStream](#)
- Overload the particle stream-related callback methods of [oapi::GraphicsClient](#), including
  - [oapi::GraphicsClient::clbkCreateParticleStream\(\)](#)
  - [oapi::GraphicsClient::clbkCreateExhaustStream\(\)](#)
  - [oapi::GraphicsClient::clbkCreateReentryStream\(\)](#)

By default, these methods return NULL pointers, i.e. don't provide particle stream support. Your overloaded methods should create an instance of an appropriate derived particle stream class and return a pointer to it.

**Important:** The client must keep track of all particle streams created. In particular, the orbiter core never deletes any particle streams it has obtained from the client. Particle stream management and cleanup must be provided by the client.

## 6.2 Attaching and detaching streams

Once a particle stream has been created, it must be connected to a vessel instance (provided by the hVessel parameter in each of the particle stream-related callback functions of the graphics client). To connect the particle stream to the vessel, use one of the [oapi::ParticleStream::Attach\(\)](#) methods using the provided vessel handle. The particle emission point and emission direction are relative to the associated vessel.

Sometimes Orbiter will call the [oapi::ParticleStream::Detach\(\)](#) method for a stream. This is usually in response to deletion of the vessel. Therefore, the stream should no longer make use of the vessel reference after it has been detached. In particular, no new particles should be generated.

**Important:** After Orbiter has detached a particle stream, it will no longer access it. The client is free to delete the particle stream instance once it has been detached. Generally, the stream should be deleted after all the remaining particles in the stream have expired.

## 6.3 Deleting streams

Generally, streams should only be deleted after they have been detached and after all remaining particles have expired. Deleting a stream with active particles will create a visual inconsistency and should be avoided. The only exception is the cleanup at the end of a simulation session.

When a stream is deleted while still attached to its object, Orbiter will call the stream's Detach method during the destruction process.

# 7 Vessel module concepts

## 7.1 Docking port management

Docking ports allow individual vessel objects to connect with each other, forming a superstructure. Orbiter automatically calculates the physical properties of the superstructure from the properties of the individual constituents. In particular, the following properties are managed by Orbiter:

- total mass: The mass of the superstructure is the sum of masses of the individual vessels
- centre of mass. The centre of mass of the superstructure is calculated from the individual vessel masses and their relative position
- inertia tensor: A simplified rigid-body model is applied to calculate an inertia tensor for the superstructure.
- effects of forces: any forces acting on individual vessels (thrust, drag, lift, etc.) are transformed into the superstructure frame and applied.

The superstructure model allows to apply the effect of forces calculated for individual vessels onto the superstructure. For example, a thrust force that acts along the centre of gravity of an individual vessel may induce a torque on the superstructure, depending on the relative position of the vessel with respect to the superstructure centre of mass.

Currently, superstructures are only supported in free flight, *not* when landed on a planetary surface. The reason is the difficulty of calculating the interaction of the composite structure with the surface. This may be addressed in the future.

## 7.2 Attachment management

Similar to docking ports, attachment points allow to connect two or more vessel objects. There are a few important differences:

- Docking ports establish peer connections, attachments establish parent-child hierarchies: A parent vessel can have multiple attached children, but each child can only be attached to a single parent.
- Attachments use a simplified physics engine: the root parent alone defines the object's trajectory (both for freespace and atmospheric flight). The children are assumed to have no influence on flight behaviour.
- Orbiter establishes docking connections automatically if the docking ports of two vessels are brought close to each other. Attachment connections are only established by API calls.
- Currently, docking connections only work in freeflight. Attachments also work for landed vessels.

Attachment connections are useful for attaching small objects to larger vessels. For example, Orbiter uses attachments to connect payload items to the Space Shuttle's cargo bay or the tip of the RMS manipulator arm (see Orbiter\SDK\samples\Atlantis).

Attachment points use an identifier string (up to 8 characters) which can provide a method to establish compatibility. For example, the Atlantis RMS arm tip will only connect to attachment points with an id string that contains "GS" in the first 2 characters (it ignores the last 6 characters). Now let's assume somebody creates another Shuttle (say a Buran) with its own RMS arm. He could then allow it to

- grapple exactly the same objects as Atlantis, by checking for "GS".
- grapple a subset of objects grappable by Atlantis, by checking additional characters, for example "GSX".
- grapple all objects grappable by Atlantis, plus additional objects, for example by checking for "GS" or "GX".
- grapple entirely different objects, for example by checking for "GX".

To connect a satellite into the payload bay, Atlantis uses the id "XS" (This means that the payload bay connection can not be used for grappling. To allow a satellite to be grappled and stored in the payload bay, it must define both a "GS" and an "XS" attachment point).

## 8 Todo List

**Member oapi::GraphicsClient::clbkGetSurfaceDC(SURFHANDLE surf)** This method should be moved into the GDIClient class

**Member oapi::GraphicsClient::clbkReleaseSurfaceDC(SURFHANDLE surf, HDC hDC)** This method should be moved into the GDIClient class

**File OrbiterAPI.h** Check functions in VESSELSTATUS2::arot and oapiGetPlanetObliquityMatrix(), minus sign has changed a place in a matrix. Is this correct??

**File OrbiterAPI.h** class CameraMode documentation

## 9 Deprecated List

**Member MFD::Update(HDC hDC)=0** This method is deprecated. MFD implementations should derive from **MFD2** and use the device-independent **MFD2::Update(oapi::Sketchpad\*)** method instead.

**Member MFD::Title(HDC hDC, const char \*title) const** This method is deprecated. MFD implementations should derive from **MFD2** and use the device-independent **MFD2::Title** method instead.

**Member MFD::SelectDefaultPen(HDC hDC, DWORD i) const** This method is deprecated. MFD implementations should derive from **MFD2** and use the device-independent **MFD2::GetDefaultPen** method instead.

**Member MFD::SelectDefaultFont(HDC hDC, DWORD i) const** This method is deprecated. MFD implementations should derive from **MFD2** and use the device-independent **MFD2::GetDefaultFont** method instead.

**Member VESSEL::GetHorizonAirspeedVector(VECTOR3 &v) const** This method has been replaced by **VESSEL::GetAirspeedVector**

**Member VESSEL::GetShipAirspeedVector(VECTOR3 &v) const** This method has been replaced by **VESSEL::GetAirspeedVector**

**Member VESSEL::SetEngineLevel(ENGINETYPE eng, double level) const** This method has been replaced by **VESSEL::SetThrusterGroupLevel**.

**Member VESSEL::IncEngineLevel(ENGINETYPE eng, double dlevel) const** This method has been replaced by **VESSEL::IncThrusterGroupLevel**.

**Member `VESSEL::SetExhaustScales(EXHAUSTTYPE exh, WORD id, double lscale, double wscale) const`**

This method no longer performs any action. It has been replaced by the `VESSEL::AddExhaust` methods.

**Member `VESSEL::DelThrusterGroup(THGROUP_HANDLE &thg, THGROUP_TYPE thgt, bool delth=false) const`**

This method has been replaced by `VESSEL::DelThrusterGroup(THGROUP_HANDLE, bool) const`.

**Member `VESSEL::GetBankMomentScale() const`** This method has been replaced by `VESSEL::GetYawMomentScale`.

**Member `VESSEL::SetBankMomentScale(double scale) const`** This method has been replaced by `VESSEL::SetYawMomentScale`.

**Member `VESSEL::SetNavRecv(DWORD n, DWORD ch) const`** This method has been replaced by `VESSEL::SetNavChannel`

**Member `VESSEL::GetNavRecv(DWORD n) const`** This method has been replaced by `VESSEL::GetNavChannel`

**Member `VESSEL::SetCOG_elev(double h) const`** This method is obsolete and should no longer be used. It has been replaced by `VESSEL::SetTouchdownPoints`.

**Member `VESSEL::ClearMeshes() const`** This version is obsolete and has been replaced by `VESSEL::ClearMeshes(bool) const`.

**Member `VESSEL::SetMeshVisibleInternal(UINT idx, bool visible) const`** This method is obsolete and has been replaced by `VESSEL::SetMeshVisibilityMode`.

**Member `VESSEL::SaveDefaultState(FILEHANDLE scn) const`** Use a call to the base class `VESSEL2::clbkSaveState` from within the overloaded callback function instead.

**Member `VESSEL::ParseScenarioLine(char *line, VESSELSTATUS *status) const`** This function is retained for backward compatibility only. New modules should overload the `VESSEL2::clbkLoadStateEx` function and use `VESSEL::ParseScenarioLineEx` for default state parsing.

**Member `VESSEL::Create(const char *name, const char *classname, const VESSELSTATUS &status)`** This method has been replaced with `oapiCreateVessel` and `oapiCreateVesselEx`.

**Member `VESSEL2::clbkDrawHUD(int mode, const HUDPAINTSPEC *hps, HDC hDC)`** This method contains a device-dependent drawing context and may not work with all graphics clients. It has been superseded by `VESSEL3::clbkDrawHUD`.

**Member VESSEL3::RegisterPanelArea(PANELHANDLE hPanel, int id, const RECT &pos, const RECT &texpos, int c)**  
This method has been superseded by [VESSEL4::RegisterPanelArea](#).

**Member oapiGetAirspeedVector** This method has been replaced by [oapiGetAirspeedVector\(OBJHANDLE,REFFRAME,VECTOR3\\*\)](#)

**Member oapiGetAtmPressureDensity** This function has been replaced by [oapiGetAtm](#).

**Member oapiGetFocusAirspeed** This method has been replaced by [oapiGetAirspeed\(\)](#)

**Member oapiGetFocusAirspeedVector** This method has been replaced by [oapiGetAirspeedVector\(OBJHANDLE,REFFRAME,VECTOR3\\*\)](#)

**Member oapiGetFocusAtmPressureDensity** This function has been replaced by [oapiGetAtm](#).

**Member oapiGetFocusShipAirspeedVector** This method has been replaced by [oapiGetAirspeedVector\(OBJHANDLE,REFFRAME,VECTOR3\\*\)](#)

**Member oapiGetMFDModeSpec** This function has been replaced by [oapiGetMFDModeSpecEx](#)

**Member oapiGetShipAirspeedVector** This method has been replaced by [oapiGetAirspeedVector\(OBJHANDLE,REFFRAME,VECTOR3\\*\)](#)

**Member oapiGetStationByIndex** Stations are no longer distinguished from vessels. This function does not perform any action other than writing a warning to the log file. Use [oapiGetVesselByIndex](#) instead.

**Member oapiGetStationByName** Stations are no longer distinguished from vessels. This function does not perform any action other than writing a warning to the log file. Use [oapiGetVesselByName](#) instead.

**Member oapiGetStationCount** Stations are no longer distinguished from vessels. This function always returns 0. Use [oapiGetVesselCount](#) instead.

**Member oapiRegisterMFDMode** This function has been replaced by [oapiRegisterMFD-Mode\(MFDMODESPECEx&\)](#).

**Member opcCloseRenderViewport** This function has been replaced by [oapi::Module::clbkSimulationEnd](#).

**Member `opcDeleteVessel`** This function has been replaced by [oapi::Module::clbkDeleteVessel](#).

**Member `opcFocusChanged`** This function has been replaced by [oapi::Module::clbkFocusChanged](#).

**Member `opcOpenRenderViewport`** This function has been replaced by [oapi::Module::clbkSimulationStart](#).

**Member `opcPause`** This function has been replaced by [oapi::Module::clbkPause](#).

**Member `opcPostStep`** This function has been replaced by [oapi::Module::clbkPostStep](#).

**Member `opcPreStep`** This function has been replaced by [oapi::Module::clbkPreStep](#).

**Member `opcTimeAccChanged`** This function has been replaced by [oapi::Module::clbkTimeAccChanged](#).

## 10 Bug List

**Member `MFD::ButtonLabel(int bt)`** This function should really return a const char\*

**Member `VESSEL::AddAnimationComponent(UINT anim, double state0, double state1, MGROUP_TRANSFORM *tr)`**

When defining a scaling transformation as a child of a parent rotation, only homogeneous scaling is supported, i.e. scale.x = scale.y = scale.z is required.

## 11 Module Index

### 11.1 Modules

Here is a list of all modules:

<b>Configuration parameter identifiers</b>	<a href="#">22</a>
<b>Render parameter identifiers</b>	<a href="#">26</a>
<b>Bit flags for planetarium mode elements</b>	<a href="#">26</a>
<b>Bit flags for blitting operations</b>	<a href="#">27</a>
<b>Some useful general constants</b>	<a href="#">28</a>
<b>Defines and Enumerations</b>	<a href="#">28</a>
<b>Ephemeris data format bitflags</b>	<a href="#">22</a>

<b>Handles</b>	<b>30</b>
<b>Surface and texture attributes</b>	<b>41</b>
<b>Mesh group editing flags</b>	<b>41</b>
<b>Bitflags for EXHAUSTSPEC flags field.</b>	<b>43</b>
<b>Light beacon shape parameters</b>	<b>44</b>
<b>Listentryflag</b>	<b>44</b>
<b>Listclbkflag</b>	<b>44</b>
<b>Animation flags</b>	<b>44</b>
<b>Identifiers for frames of reference</b>	<b>45</b>
<b>Thruster and thruster-group parameters</b>	<b>45</b>
<b>Airfoil orientation</b>	<b>46</b>
<b>Aerodynamic control surface types</b>	<b>47</b>
<b>Control surface axis orientation</b>	<b>47</b>
<b>Identifiers for visual events</b>	<b>48</b>
<b>Navigation mode identifiers</b>	<b>49</b>
<b>Manual control mode identifiers</b>	<b>49</b>
<b>Manual control device identifiers</b>	<b>50</b>
<b>RCS mode identifiers</b>	<b>50</b>
<b>HUD mode identifiers</b>	<b>51</b>
<b>MFD mode identifiers</b>	<b>51</b>
<b>MFD identifiers</b>	<b>52</b>
<b>Panel neighbour identifiers</b>	<b>53</b>
<b>Panel redraw event identifiers</b>	<b>54</b>
<b>Mouse event identifiers</b>	<b>54</b>
<b>Panel area texture mapping identifiers</b>	<b>55</b>
<b>Generic vessel message identifiers</b>	<b>56</b>
<b>Vessel mesh visibility flags</b>	<b>56</b>
<b>Navigation radio transmitter types</b>	<b>57</b>
<b>Object parameter flags</b>	<b>57</b>

<b>Keyboard key identifiers</b>	<b>186</b>
<b>Logical key ids</b>	<b>192</b>
<b>Structure definitions</b>	<b>29</b>
<b>Identifiers for special render surfaces</b>	<b>31</b>
<b>Vectors and matrices</b>	<b>31</b>
<b>Local lighting interface</b>	<b>43</b>
<b>Orbiter API interface methods</b>	<b>58</b>
<b>Object access functions</b>	<b>65</b>
<b>Vessel creation and destruction</b>	<b>73</b>
<b>Body functions</b>	<b>75</b>
<b>Vessel functions</b>	<b>77</b>
<b>Coordinate transformations</b>	<b>92</b>
<b>Camera functions</b>	<b>96</b>
<b>Functions for planetary bodies</b>	<b>102</b>
<b>Surface base interface</b>	<b>109</b>
<b>Time functions</b>	<b>112</b>
<b>Navigation radio transmitter functions</b>	<b>116</b>
<b>Script interpreter functions</b>	<b>120</b>
<b>Visual and mesh functions</b>	<b>122</b>
<b>HUD, MFD and panel functions</b>	<b>133</b>
<b>Drawing support functions</b>	<b>146</b>
<b>Surface functions</b>	<b>151</b>
<b>Custom MFD mode definition</b>	<b>156</b>
<b>Virtual cockpit functions</b>	<b>159</b>
<b>Customisation - custom menu, dialogs</b>	<b>163</b>
<b>File IO Functions</b>	<b>170</b>
<b>Utility functions</b>	<b>179</b>
<b>User input functions</b>	<b>180</b>
<b>Onscreen annotations</b>	<b>180</b>

<b>Obsolete functions</b>	<b>183</b>
<b>Top-level module callback functions</b>	<b>199</b>
<b>General module callback functions</b>	<b>199</b>
<b>Vessel module callback functions</b>	<b>200</b>
<b>Plugin module callback functions</b>	<b>201</b>

## 12 Class Index

### 12.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

<b>ANIMATION</b>	<b>206</b>
<b>ANIMATIONCOMP</b>	<b>207</b>
<b>ATMCONST</b>	<b>207</b>
<b>ATMOSPHERE</b>	<b>209</b>
<b>ATMOSPHERE::PRM_IN</b>	<b>211</b>
<b>ATMOSPHERE::PRM_OUT</b>	<b>212</b>
<b>ATMPARAM</b>	<b>213</b>
<b>BEACONLIGHTSPEC</b>	<b>213</b>
<b>CELBODY</b>	<b>216</b>
<b>CELBODY2</b>	<b>220</b>
<b>COLOUR4</b>	<b>225</b>
<b>oapi::DrawingTool</b>	<b>226</b>
<b>oapi::Brush</b>	<b>215</b>
<b>oapi::Font</b>	<b>234</b>
<b>oapi::Pen</b>	<b>328</b>
<b>ELEMENTS</b>	<b>226</b>
<b>ENGINESTATUS</b>	<b>227</b>
<b>EXHAUSTSPEC</b>	<b>228</b>
<b>ExternMFD</b>	<b>229</b>
<b>FogParam</b>	<b>233</b>

<b>oapi::GraphicsClient::LABELLIST</b>	<b>275</b>
<b>oapi::GraphicsClient::VIDEODATA</b>	<b>275</b>
<b>GROUPEDITSPEC</b>	<b>280</b>
<b>HELPCONTEXT</b>	<b>281</b>
<b>HUDPARAM</b>	<b>282</b>
<b>oapi::IVECTOR2</b>	<b>282</b>
<b>LaunchpadItem</b>	<b>283</b>
<b>LightEmitter</b>	<b>286</b>
<b>PointLight</b>	<b>329</b>
<b>SpotLight</b>	<b>347</b>
<b>LISTENTRY</b>	<b>292</b>
<b>MATERIAL</b>	<b>292</b>
<b>MATRIX3</b>	<b>293</b>
<b>MESHGROUP</b>	<b>294</b>
<b>MESHGROUP_TRANSFORM</b>	<b>295</b>
<b>MESHGROUPEX</b>	<b>296</b>
<b>MFD</b>	<b>297</b>
<b>GraphMFD</b>	<b>276</b>
<b>MFD2</b>	<b>305</b>
<b>oapi::ModuleNV</b>	<b>315</b>
<b>oapi::Module</b>	<b>310</b>
<b>oapi::GraphicsClient</b>	<b>236</b>
<b>NAVDATA</b>	<b>318</b>
<b>NTVERTEX</b>	<b>319</b>
<b>ORBITPARAM</b>	<b>320</b>
<b>oapi::ParticleStream</b>	<b>321</b>
<b>PARTICLESTREAMSPEC</b>	<b>326</b>
<b>oapi::ScreenAnnotation</b>	<b>332</b>
<b>oapi::Sketchpad</b>	<b>335</b>

VECTOR3	350
VESSEL	351
VESSEL2	503
VESSEL3	520
VESSEL4	530
VESSELSTATUS	531
VESSELSTATUS2	533
VESSELSTATUS2::DOCKINFOSPEC	537
VESSELSTATUS2::FUELSPEC	537
VESSELSTATUS2::THRUSTSPEC	538

## 13 Class Index

### 13.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<b>ANIMATION</b> (Animation definition )	206
<b>ANIMATIONCOMP</b> (Animation component definition )	207
<b>ATMCONST</b> (Planetary atmospheric constants structure )	207
<b>ATMOSPHERE</b> (Defines the physical atmospheric properties for a celestial body )	209
<b>ATMOSPHERE::PRM_IN</b> (Input parameters for atmospheric data calculation )	211
<b>ATMOSPHERE::PRM_OUT</b> (Output parameters for atmospheric data calculation )	212
<b>ATMPARAM</b> (Atmospheric parameters structure )	213
<b>BEACONLIGHTSPEC</b> (Vessel beacon light parameters )	213
<b>oapi::Brush</b> (A brush is a drawing resource for filling closed figures (rectangles, ellipses, polygons) )	215
<b>CELBODY</b> (This is the base class for celestial body classes )	216
<b>CELBODY2</b> (Extension to <b>CELBODY</b> class )	220
<b>COLOUR4</b> (Colour definition )	225
<b>oapi::DrawingTool</b> (Base class for various 2-D drawing resources (fonts, pens, brushes, etc.) )	226
<b>ELEMENTS</b> (Kepler orbital elements )	226
<b>ENGINESTATUS</b> (Engine status )	227

<b>EXHAUSTSPEC</b> (Engine exhaust render parameters )	228
<b>ExternMFD</b> (ExternMFD provides support for defining an MFD display in a plugin module)	229
<b>FogParam</b> (Distance fog render parameters )	233
<b>oapi::Font</b> (A font resource for drawing text. A font has a defined size, typeface, slant, weight, etc. Fonts can be selected into a Sketchpad and then apply to all subsequent Text calls )	234
<b>oapi::GraphicsClient</b> (Base class for external graphics client modules )	236
<b>oapi::GraphicsClient::LABELLIST</b> (Label list description for celestial and surface markers )	275
<b>oapi::GraphicsClient::VIDEODATA</b> (Structure containing default video options, as stored in Orbiter.cfg )	275
<b>GraphMFD</b> (This class is derived from MFD and provides a template for MFD modes containing 2D graphs )	276
<b>GROUPEDITSPEC</b> (Structure used by oapiEditMeshGroup to define the group elements to be replaced )	280
<b>HELPCONTEXT</b> (Context information for an Orbiter ingame help page )	281
<b>HUDPARAM</b> (Mode-specific parameters for HUD mode settings )	282
<b>oapi::IVECTOR2</b> (Integer-valued 2-D vector type )	282
<b>LaunchpadItem</b> (Base class to define launchpad items )	283
<b>LightEmitter</b> (Base class for defining a light source that can illuminate other objects )	286
<b>LISTENTRY</b> (Entry specification for selection list entry )	292
<b>MATERIAL</b> (Material definition )	292
<b>MATRIX3</b> (3x3-element matrix )	293
<b>MESHGROUP</b> (Defines a mesh group (subset of a mesh) )	294
<b>MESHGROUP_TRANSFORM</b> (This structure defines an affine mesh group transform (translation, rotation or scaling) )	295
<b>MESHGROUPEX</b> (Extended mesh group definition )	296
<b>MFD</b> (This class acts as an interface for user defined MFD (multi functional display) modes)	297
<b>MFD2</b> (Extended MFD class )	305
<b>oapi::Module</b> (Generic Orbiter plugin interface class )	310
<b>oapi::ModuleNV</b> (Generic Orbiter plugin interface class )	315
<b>NAVDATA</b> (Navigation transmitter data )	318
<b>NTVERTEX</b> (Vertex definition including normals and texture coordinates )	319

<b>ORBITPARAM</b> (Secondary orbital parameters derived from the primary <b>ELEMENTS</b> )	320
<b>oapi::ParticleStream</b> (Defines an array of "particles" (e.g. for exhaust and reentry effects, gas venting, condensation, etc.) )	321
<b>PARTICLESTREAMSPEC</b> (Particle stream parameters )	326
<b>oapi::Pen</b> (A pen is a resource used for drawing lines and the outlines of closed figures such as rectangles, ellipses and polygons )	328
<b>PointLight</b> (Class for isotropic point light source )	329
<b>oapi::ScreenAnnotation</b> (Defines a block of text displayed on top of the simulation render window )	332
<b>oapi::Sketchpad</b> (A Sketchpad object defines an environment for drawing onto 2-D surfaces)	335
<b>SpotLight</b> (Class for directed spot light sources )	347
<b>VECTOR3</b> (3-element vector )	350
<b>VESSEL</b> (Base class for objects of vessel type (spacecraft and similar) )	351
<b>VESSEL2</b> (Callback extensions to the <b>VESSEL</b> class )	503
<b>VESSEL3</b> (Callback extensions to the <b>VESSEL</b> class )	520
<b>VESSEL4</b> (Extensions to the <b>VESSEL</b> class )	530
<b>VESSELSTATUS</b> (Vessel status parameters (version 1) )	531
<b>VESSELSTATUS2</b> (Vessel status parameters (version 2) )	533
<b>VESSELSTATUS2::DOCKINFOSPEC</b> (Dock info list )	537
<b>VESSELSTATUS2::FUELSPEC</b> (Propellant list )	537
<b>VESSELSTATUS2::THRUSTSPEC</b> (Thruster definition list )	538

## 14 File Index

### 14.1 File List

Here is a list of all documented files with brief descriptions:

<b>Orbitersdk/include/CelBodyAPI.h</b> (Contains interface classes for celestial bodies: <b>CELBODY</b> and <b>CELBODY2</b> )	538
<b>Orbitersdk/include/DrawAPI.h</b> (2-D surface drawing support interface )	539
<b>Orbitersdk/include/MFDAPI.h</b> (Class interfaces for <b>MFD</b> instruments and <b>MFD</b> modes )	540
<b>Orbitersdk/include/OrbiterAPI.h</b> (General API interface functions )	541

Orbitersdk/include/[VesselAPI.h](#) (Contains the class interfaces for vessel objects (**VESSEL**, **VESSEL2**, **VESSEL3**) ) 592

## 15 Module Documentation

### 15.1 Ephemeris data format bitflags

#### 15.1.1 Detailed Description

Ephemeris data format bitflags

#### Defines

- #define **EPHEM\_TRUEPOS** 0x01  
*true body position*
- #define **EPHEM\_TRUEVEL** 0x02  
*true body velocity*
- #define **EPHEM\_BARYPOS** 0x04  
*barycentric position*
- #define **EPHEM\_BARYVEL** 0x08  
*barycentric velocity*
- #define **EPHEM\_BARYISTRUE** 0x10  
*body has no child objects*
- #define **EPHEM\_PARENTBARY** 0x20  
*ephemerides are computed in terms of the barycentre of the parent body's system*
- #define **EPHEM\_POLAR** 0x40  
*data is returned in polar format*

### 15.2 Configuration parameter identifiers

#### 15.2.1 Detailed Description

Used by GraphicsClient::GetConfigParam()

#### Defines

- #define **CFGPRM\_SURFACEMAXLEVEL** 0x0001
- #define **CFGPRM\_SURFACEREFLECT** 0x0002
- #define **CFGPRM\_SURFACERIPPLE** 0x0003
- #define **CFGPRM\_SURFACELIGHTS** 0x0004
- #define **CFGPRM\_SURFACELIGHTBRT** 0x0006

- #define `CFGPRM_ATMHAZE` 0x0007
- #define `CFGPRM_ATMFOG` 0x0008
- #define `CFGPRM_CLOUDS` 0x0009
- #define `CFGPRM_CLOUDSHADOWS` 0x000A
- #define `CFGPRM_PLANETARIUMFLAG` 0x000B
- #define `CFGPRM_STARRENDERPRM` 0x000C
- #define `CFGPRM_AMBIENTLEVEL` 0x000E
- #define `CFGPRM_VESSELSHADOWS` 0x000F
- #define `CFGPRM_OBJECTSHADOWS` 0x0010
- #define `CFGPRM_OBJECTSPECULAR` 0x0011
- #define `CFGPRM_CSHERETEXTURE` 0x0013
- #define `CFGPRM_CSHEREINTENS` 0x0014
- #define `CFGPRM_LOCALLIGHT` 0x0015
- #define `CFGPRM_MAXLIGHT` 0x0016

### 15.2.2 Define Documentation

#### 15.2.2.1 #define `CFGPRM_AMBIENTLEVEL` 0x000E

Ambient light level (brightness of unlit nonemissive surfaces) in the range 0-255.

**Parameter type:**

DWORD

#### 15.2.2.2 #define `CFGPRM_ATMFOG` 0x0008

Flag for rendering distance fog within planetary atmospheres

**Parameter type:**

bool

#### 15.2.2.3 #define `CFGPRM_ATMHAZE` 0x0007

Flag for rendering "atmospheric haze" over planets with atmospheres

**Parameter type:**

bool

#### 15.2.2.4 #define `CFGPRM_CLOUDS` 0x0009

Flag for rendering planetary cloud layers

**Parameter type:**

bool

**15.2.2.5 #define CFGPRM\_CLOUDSHADOWS 0x000A**

Flag for rendering cloud shadows on the planet surface

**Parameter type:**

bool

**15.2.2.6 #define CFGPRM\_CSPHEREINTENS 0x0014**

Flag for rendering intensity of celestial sphere background textures

**Parameter type:**

double

**15.2.2.7 #define CFGPRM\_CSPHERETEXTURE 0x0013**

File path for celestial sphere background textures

**Parameter type:**

\*char

**15.2.2.8 #define CFGPRM\_LOCALLIGHT 0x0015**

Flag for enabling local light sources

**Parameter type:**

bool

**15.2.2.9 #define CFGPRM\_MAXLIGHT 0x0016**

Max number of light sources

**Parameter type:**

int

**15.2.2.10 #define CFGPRM\_OBJECTSHADOWS 0x0010**

Flag for rendering object shadows on the planet surface

**Parameter type:**

bool

**15.2.2.11 #define CFGPRM\_OBJECTSPECULAR 0x0011**

Flag for enabling specular reflections from objects

**Parameter type:**

bool

**15.2.2.12 #define CFGPRM\_PLANETARIUMFLAG 0x000B**

Bit flag for "planetarium mode" elements. For a description of the available bit flags, see [Bit flags for planetarium mode elements](#)

**Parameter type:**

DWORD

**15.2.2.13 #define CFGPRM\_STARRENDERPRM 0x000C**

Parameters for rendering stars on the celestial sphere.

**Parameter type:**

struct StarRenderPrm

**15.2.2.14 #define CFGPRM\_SURFACELIGHTBRT 0x0006**

Brightness factor for emissive city light textures (0-1)

**Parameter type:**

double

**15.2.2.15 #define CFGPRM\_SURFACELIGHTS 0x0004**

Flag for rendering emissive textures ("city lights") on the unlit side of planetary surfaces.

**Parameter type:**

bool

**15.2.2.16 #define CFGPRM\_SURFACEMAXLEVEL 0x0001**

Max. level of detail for rendering planetary surfaces (1-10)

**Parameter type:**

DWORD

**15.2.2.17 #define CFGPRM\_SURFACEREFLECT 0x0002**

Flag for rendering planet surfaces with specular reflections (e.g. for oceans)

**Parameter type:**

bool

**15.2.2.18 #define CFGPRM\_SURFACERIPPLE 0x0003**

Flag for rendering specular reflections with microtexture ("ripples")

**Parameter type:**

bool

**15.2.2.19 #define CFGPRM\_VESSELSHADOWS 0x000F**

Flag for rendering vessel shadows on the planet surface

**Parameter type:**

bool

## 15.3 Render parameter identifiers

### 15.3.1 Detailed Description

**See also:**

GraphicsClient::clbkGetRenderParam()

**Defines**

- #define RP\_COLOURDEPTH 1  
*colour bit depth of the render target*
- #define RP\_ZBUFFERDEPTH 2  
*z-buffer depth of the render target*
- #define RP\_STENCILDEPTH 3  
*stencil buffer depth of the render target*
- #define RP\_MAXLIGHTS 4  
*maximum number of simultaneous light sources supported*
- #define RP\_ISTLDEVICE 5

### 15.3.2 Define Documentation

#### 15.3.2.1 #define RP\_ISTLDEVICE 5

render device supports Transform&Lighting

## 15.4 Bit flags for planetarium mode elements

**Defines**

- #define PLN\_ENABLE 0x0001  
*Enable planetarium mode (master flag).*
- #define PLN\_CGRID 0x0002  
*Enable celestial grid.*
- #define PLN\_EGRID 0x0004  
*Enable ecliptic grid.*
- #define PLN\_ECL 0x0008

*Enable line of ecliptic.*

- #define **PLN\_EQU** 0x0010

*Enable celestial equator.*

- #define **PLN\_CONST** 0x0020

*Enable constellation lines.*

- #define **PLN\_CNSTLABEL** 0x0040

*Enable constellation labels.*

- #define **PLN\_CMARK** 0x0080

*Enable celestial body markers.*

- #define **PLN\_VMARK** 0x0100

*Enable vessel markers.*

- #define **PLN\_BMARK** 0x0200

*Enable surface base markers.*

- #define **PLN\_RMARK** 0x0400

*Enable VOR transmitter markers.*

- #define **PLN\_LMARK** 0x0800

*Enable planetary surface labels.*

- #define **PLN\_CNSTLONG** 0x1000

*Enable long constellation names.*

- #define **PLN\_CNSTSHORT** 0x2000

*Enable short constellation names.*

- #define **PLN\_CCMARK** 0x4000

*Enable celestial sphere labels.*

- #define **PLN\_SURFMARK** (PLN\_BMARK | PLN\_RMARK | PLN\_LMARK)

## 15.5 Bit flags for blitting operations

### Defines

- #define **BLT\_SRCCOLORKEY** 0x1

*Use source surface colour key for transparency.*

- #define **BLT\_TGTCOLORKEY** 0x2

### 15.5.1 Define Documentation

#### 15.5.1.1 #define **BLT\_TGTCOLORKEY** 0x2

Use target surface colour key for transparency

## 15.6 Some useful general constants

### Variables

- const double **PI** = 3.14159265358979323846  
*pi*
- const double **PI05** = 1.57079632679489661923  
*pi/2*
- const double **PI2** = 6.28318530717958647693  
*pi\*2*
- const double **RAD** = **PI**/180.0  
*factor to map degrees to radians*
- const double **DEG** = 180.0/**PI**  
*factor to map radians to degrees*
- const double **C0** = 299792458.0  
*speed of light in vacuum [m/s]*
- const double **TAUA** = 499.004783806  
*light time for 1 AU [s]*
- const double **AU** = **C0**\***TAUA**  
*astronomical unit (mean geocentric distance of the sun) [m]*
- const double **GGRAV** = 6.67259e-11  
*gravitational constant [m^3 kg^-1 s^-2]*
- const double **G** = 9.81  
*gravitational acceleration [m/s^2] at Earth mean radius*
- const double **ATMP** = 101.4e3  
*atmospheric pressure [Pa] at Earth sea level*
- const double **ATMD** = 1.293  
*atmospheric density [kg/m^3] at Earth sea level*

## 15.7 Defines and Enumerations

### Modules

- Ephemeris data format bitflags
- Handles
- Surface and texture attributes
- Mesh group editing flags
- Bitflags for EXHAUSTSPEC flags field.

- Light beacon shape parameters
- Listentryflag
- Listclbkflag
- Animation flags
- Identifiers for frames of reference
- Thruster and thruster-group parameters
- Airfoil orientation
- Aerodynamic control surface types
- Control surface axis orientation
- Identifiers for visual events
- Navigation mode identifiers
- Manual control mode identifiers
- Manual control device identifiers
- RCS mode identifiers
- HUD mode identifiers
- MFD mode identifiers
- MFD identifiers
- Panel neighbour identifiers
- Panel redraw event identifiers
- Mouse event identifiers
- Panel area texture mapping identifiers
- Generic vessel message identifiers
- Vessel mesh visibility flags
- Navigation radio transmitter types
- Object parameter flags
- Keyboard key identifiers
- Logical key ids

## 15.8 Structure definitions

### Classes

- struct **COLOUR4**  
*colour definition*
- struct **NTVERTEX**  
*vertex definition including normals and texture coordinates*
- struct **MESHGROUP**  
*Defines a mesh group (subset of a mesh).*
- struct **MESHGROUPEX**  
*extended mesh group definition*
- struct **GROUPEDITSPEC**  
*Structure used by [oapiEditMeshGroup](#) to define the group elements to be replaced.*
- struct **MATERIAL**  
*material definition*

- struct **ATMCONST**  
*Planetary atmospheric constants structure.*

## 15.9 Handles

### Typedefs

- typedef void \* **OBJHANDLE**  
*Handle for objects (vessels, stations, planets).*
- typedef void \* **VISHANDLE**  
*Handle for visuals.*
- typedef void \* **MESHHANDLE**  
*Handle for meshes.*
- typedef int \* **DEVMESHHANDLE**  
*Handle for graphics-client-specific meshes.*
- typedef void \* **SURFHANDLE**  
*Handle for bitmap surfaces and textures (panels and panel items).*
- typedef void \* **PANELHANDLE**  
*Handle for 2D instrument panels.*
- typedef void \* **FILEHANDLE**  
*Handle for file streams.*
- typedef void \* **INTERPRETERHANDLE**  
*Handle for script interpreters.*
- typedef void \* **THRUSTER\_HANDLE**  
*Handle for thrusters.*
- typedef void \* **THGROUP\_HANDLE**  
*Handle for logical thruster groups.*
- typedef void \* **PROPELLANT\_HANDLE**  
*Propellant resource handle.*
- typedef void \* **PSTREAM\_HANDLE**  
*Handle for particle streams.*
- typedef void \* **DOCKHANDLE**  
*Handle for vessel docking ports.*
- typedef void \* **ATTACHMENTHANDLE**  
*Handle for vessel passive attachment points.*

- `typedef void * AIRFOILHANDLE`  
*Handle for vessel airfoils.*
- `typedef void * CTRLSURFHANDLE`  
*Handle for vessel aerodynamic control surfaces.*
- `typedef void * NAVHANDLE`  
*Handle for a navigation radio transmitter (VOR, ILS, IDS, XPDR).*
- `typedef void * ANIMATIONCOMPONENT_HANDLE`  
*Handle for animation components.*
- `typedef void * LAUNCHPADITEM_HANDLE`  
*Handle for custom items added to Launchpad "Extra" list.*
- `typedef void * NOTEHANDLE`  
*Handle for onscreen annotation objects.*

## 15.10 Identifiers for special render surfaces

### Defines

- `#define RENDERTGT_NONE ((SURFHANDLE)-1)`  
*no surface*
- `#define RENDERTGT_MAINWINDOW 0`

#### 15.10.1 Define Documentation

##### 15.10.1.1 #define RENDERTGT\_MAINWINDOW 0

main render target

## 15.11 Vectors and matrices

### 15.11.1 Detailed Description

Vectors and matrices are used to represent positions, velocities, translations, rotations, etc. in the 3-dimensional object space. Orbiter provides the VECTOR3 and MATRIX3 structures for 3-D vectors and matrices. A number of utility functions allow common operations such as matrix-vector products, dot and vector products, etc.

### Classes

- union `VECTOR3`  
*3-element vector*
- union `MATRIX3`  
*3x3-element matrix*

## Functions

- **VECTOR3 \_V** (double x, double y, double z)  
*Vector composition.*
- void **veccpy** (VECTOR3 &a, const VECTOR3 &b)  
*Vector copy.*
- **VECTOR3 operator+** (const VECTOR3 &a, const VECTOR3 &b)  
*Vector addition.*
- **VECTOR3 operator-** (const VECTOR3 &a, const VECTOR3 &b)  
*Vector subtraction.*
- **VECTOR3 operator\*** (const VECTOR3 &a, const double f)  
*Multiplication of vector with scalar.*
- **VECTOR3 operator/** (const VECTOR3 &a, const double f)  
*Division of vector by a scalar.*
- **VECTOR3 & operator+=** (VECTOR3 &a, const VECTOR3 &b)  
*Vector addition-assignment  $a += b$ .*
- **VECTOR3 & operator-=** (VECTOR3 &a, const VECTOR3 &b)  
*Vector subtraction-assignment  $a -= b$ .*
- **VECTOR3 & operator\*=**(VECTOR3 &a, const double f)****  
*Vector-scalar multiplication-assignment  $a *= f$ .*
- **VECTOR3 & operator/=**(VECTOR3 &a, const double f)****  
*Vector-scalar division-assignment  $a /= f$ .*
- **VECTOR3 operator-** (const VECTOR3 &a)  
*Vector unary minus  $-a$ .*
- double **dotp** (const VECTOR3 &a, const VECTOR3 &b)  
*Scalar (inner, dot) product of two vectors.*
- **VECTOR3 crossp** (const VECTOR3 &a, const VECTOR3 &b)  
*Vector (cross) product of two vectors.*
- double **length** (const VECTOR3 &a)  
*Length (L2-norm) of a vector.*
- double **dist** (const VECTOR3 &a, const VECTOR3 &b)  
*Distance between two points.*
- void **normalise** (VECTOR3 &a)  
*Normalise a vector.*

- **VECTOR3 unit** (const VECTOR3 &a)  
*Returns normalised vector.*
- **MATRIX3 \_M** (double m11, double m12, double m13, double m21, double m22, double m23, double m31, double m32, double m33)  
*Matrix composition.*
- **MATRIX3 identity** ()  
*Returns the identity matrix.*
- **MATRIX3 outerp** (const VECTOR3 &a, const VECTOR3 &b)  
*Outer product of two vectors.*
- **MATRIX3 operator+** (const MATRIX3 &A, double s)  
*Sum of matrix and scalar.*
- **MATRIX3 operator-** (const MATRIX3 &A, double s)  
*Difference of matrix and scalar.*
- **MATRIX3 operator\*** (const MATRIX3 &A, double s)  
*Product of matrix and scalar.*
- **MATRIX3 operator/** (const MATRIX3 &A, double s)  
*Quotient of matrix and scalar.*
- **MATRIX3 & operator\*=(** MATRIX3 &A, double s)  
*Matrix-scalar product-assignment A \*= s.*
- **MATRIX3 & operator/=(** MATRIX3 &A, double s)  
*Matrix-scalar division-assignment A /= s.*
- **VECTOR3 mul** (const MATRIX3 &A, const VECTOR3 &b)  
*Matrix-vector multiplication.*
- **VECTOR3 tmul** (const MATRIX3 &A, const VECTOR3 &b)  
*Matrix transpose-vector multiplication.*
- **MATRIX3 mul** (const MATRIX3 &A, const MATRIX3 &B)  
*Matrix-matrix multiplication.*

### 15.11.2 Function Documentation

#### 15.11.2.1 MATRIX3 \_M (double m11, double m12, double m13, double m21, double m22, double m23, double m31, double m32, double m33) [inline]

Matrix composition.

Returns a matrix composed of the provided elements.

**Returns:**

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix}$$

**15.11.2.2 VECTOR3\_V (double *x*, double *y*, double *z*) [inline]**

Vector composition.

Returns a vector composed of the three provided arguments

**Parameters:**

*x* x-component

*y* y-component

*z* z-component

**Returns:**

vector defined as (x,y,z)

**Examples:**

[clbkPreStep.cpp](#).

**15.11.2.3 VECTOR3 crossp (const VECTOR3 & *a*, const VECTOR3 & *b*) [inline]**

Vector (cross) product of two vectors.

**Parameters:**

$\leftarrow \mathbf{a}$  First vector operand

$\leftarrow \mathbf{b}$  Second vector operand

**Returns:**

Vector product **axb**

**15.11.2.4 double dist (const VECTOR3 & *a*, const VECTOR3 & *b*) [inline]**

Distance between two points.

**Parameters:**

$\leftarrow \mathbf{a}$  First point

$\leftarrow \mathbf{b}$  Second point

**Returns:**

Distance between a and b

**15.11.2.5 double dotp (const VECTOR3 & *a*, const VECTOR3 & *b*) [inline]**

Scalar (inner, dot) product of two vectors.

**Parameters:**

- ← *a* First vector operand
- ← *b* Second vector operand

**Returns:**

Scalar product **ab**

**15.11.2.6 double length (const VECTOR3 & *a*) [inline]**

Length (L2-norm) of a vector.

**Parameters:**

- *a* Vector operand

**Returns:**

Vector norm  $|\mathbf{a}|_2$

**15.11.2.7 MATRIX3 mul (const MATRIX3 & *A*, const MATRIX3 & *B*) [inline]**

Matrix-matrix multiplication.

**Parameters:**

- ← *A* First matrix operand
- ← *B* Second matrix operand

**Returns:**

Result of **AB**

**15.11.2.8 VECTOR3 mul (const MATRIX3 & *A*, const VECTOR3 & *b*) [inline]**

Matrix-vector multiplication.

**Parameters:**

- ← *A* matrix operand
- ← *b* vector operand

**Returns:**

Result of **Ab**

**15.11.2.9 void normalise (VECTOR3 & *a*) [inline]**

Normalise a vector.

Resizes the argument vector to length 1.

**Parameters:**

$\leftrightarrow a$  Vector argument

**Note:**

The length of *a* must be greater than 0.

**15.11.2.10 MATRIX3 operator\* (const MATRIX3 & *A*, double *s*) [inline]**

Product of matrix and scalar.

**Parameters:**

$\leftarrow A$  Matrix operand (left)

$\leftarrow s$  scalar operand (right)

**Returns:**

*A\*s* (element-wise product of *A* and *s*)

**15.11.2.11 VECTOR3 operator\* (const VECTOR3 & *a*, const double *f*) [inline]**

Multiplication of vector with scalar.

**Parameters:**

*a* vector operand

*f* scalar operand

**Returns:**

Result of element-wise *a\*f*.

**15.11.2.12 MATRIX3& operator\*=(MATRIX3 & *A*, double *s*) [inline]**

Matrix-scalar product-assignment *A \*= s*.

**Parameters:**

$\leftarrow A$  Matrix operand (left)

$\leftarrow s$  scalar operand (right)

**Returns:**

Replaces *A* with element-wise product *A\*s* and returns the result.

**15.11.2.13 VECTOR3& operator\*=(VECTOR3 & a, const double f) [inline]**

Vector-scalar multiplication-assignment a \*= f.

**Parameters:**

- ↔ *a* Left-hand vector operand
- ← *f* Right hand scalar operand

**Returns:**

Replaces a with element-wise a\*f and returns the result.

**15.11.2.14 MATRIX3 operator+ (const MATRIX3 & A, double s) [inline]**

Sum of matrix and scalar.

**Parameters:**

- ← *A* Matrix operand (left)
- ← *s* scalar operand (right)

**Returns:**

A+s (element-wise sum of A and s)

**15.11.2.15 VECTOR3 operator+ (const VECTOR3 & a, const VECTOR3 & b) [inline]**

Vector addition.

**Parameters:**

- a* first vector operand
- b* second vector operand

**Returns:**

Result of a+b.

**15.11.2.16 VECTOR3& operator+= (VECTOR3 & a, const VECTOR3 & b) [inline]**

Vector addition-assignment a += b.

**Parameters:**

- ↔ *a* Left-hand vector operand
- ← *b* Right-hand vector operand

**Returns:**

Replaces a with a+b and returns the result.

**15.11.2.17 MATRIX3 operator- (const MATRIX3 & A, double s) [inline]**

Difference of matrix and scalar.

**Parameters:**

- ← *A* Matrix operand (left)
- ← *s* scalar operand (right)

**Returns:**

*A*-*s* (element-wise difference of *A* and *s*)

**15.11.2.18 VECTOR3 operator- (const VECTOR3 & a) [inline]**

Vector unary minus -*a*.

**Parameters:**

- ← *a* Vector operand

**Returns:**

Negative vector (-*a*.x, -*a*.y, -*a*.z)

**15.11.2.19 VECTOR3 operator- (const VECTOR3 & a, const VECTOR3 & b) [inline]**

Vector subtraction.

**Parameters:**

- a* first vector operand
- b* second vector operand

**Returns:**

Result of *a*-*b*.

**15.11.2.20 VECTOR3& operator-= (VECTOR3 & a, const VECTOR3 & b) [inline]**

Vector subtraction-assignment *a* -= *b*.

**Parameters:**

- ↔ *a* Left-hand vector operand
- ← *b* Right-hand vector operand

**Returns:**

Replaces *a* with *a*-*b* and returns the result.

**15.11.2.21 MATRIX3 operator/ (const MATRIX3 & A, double s) [inline]**

Quotient of matrix and scalar.

**Parameters:**

$\leftarrow A$  Matrix operand (left)

$\leftarrow s$  scalar operand (right)

**Returns:**

A/s (element-wise quotient of A and s)

**Note:**

s != 0 is required.

**15.11.2.22 VECTOR3 operator/ (const VECTOR3 & a, const double f) [inline]**

Division of vector by a scalar.

**Parameters:**

$a$  vector operand

$f$  scalar operand

**Returns:**

Result of element-wise a/f.

**15.11.2.23 MATRIX3& operator/= (MATRIX3 & A, double s) [inline]**

Matrix-scalar division-assignment A /= s.

**Parameters:**

$\leftarrow A$  Matrix operand (left)

$\leftarrow s$  scalar operand (right)

**Returns:**

Replaces A with element-wise quotient A/s and returns the result.

**Note:**

s != 0 is required.

**15.11.2.24 VECTOR3& operator/= (VECTOR3 & a, const double f) [inline]**

Vector-scalar division-assignment a /= f.

**Parameters:**

$\leftrightarrow a$  Left-hand vector operand

$\leftarrow f$  Right-hand scalar operand

**Returns:**

Replaces a with element-wise a/f and returns the result.

**15.11.2.25 MATRIX3 outerp (const VECTOR3 & *a*, const VECTOR3 & *b*) [inline]**

Outer product of two vectors.

**Parameters:**

- ← *a* First vector operand
- ← *b* Second vector operand

**Returns:**

Outer product  $\mathbf{ab}^T$ , where **a** and **b** represent column vectors.

**15.11.2.26 VECTOR3 tmul (const MATRIX3 & *A*, const VECTOR3 & *b*) [inline]**

Matrix transpose-vector multiplication.

**Parameters:**

- ← *A* matrix operand
- ← *b* vector operand

**Returns:**

Result of  $\mathbf{A}^T \mathbf{b}$

**15.11.2.27 VECTOR3 unit (const VECTOR3 & *a*) [inline]**

Returns normalised vector.

Returns a vector of length 1 with the same direction as the argument vector.

**Parameters:**

- ← *a* Vector argument

**Returns:**

Normalised vector.

**Note:**

The length of *a* must be greater than 0.

**15.11.2.28 void veccp (VECTOR3 & *a*, const VECTOR3 & *b*) [inline]**

Vector copy.

Copies the element values from the source to the target vector.

**Parameters:**

- *a* target vector
- ← *b* source vector

## 15.12 Surface and texture attributes

### 15.12.1 Detailed Description

These bitflags are used during texture creation or loading to specify what they will be used for. This information is required by the graphics clients to initialise and optimise the surfaces accordingly.

**See also:**

`oapiCreateSurfaceEx`, `oapiLoadSurfaceEx`

#### Defines

- `#define OAPISURFACE_TEXTURE 0x0001`  
*Surface can be used as a texture (e.g. by associating it with a mesh).*
- `#define OAPISURFACE_RENDERTARGET 0x0002`  
*Surface can be rendered to by the graphics device.*
- `#define OAPISURFACE_GDI 0x0004`  
*A HDC context can be requested from the surface for GDI drawing.*
- `#define OAPISURFACE_SKETCHPAD 0x0008`  
*A Sketchpad context can be requested from the surface for Sketchpad drawing.*
- `#define OAPISURFACE_MIPMAPS 0x0010`  
*Create a full chain of mipmaps for the surface. If loaded from file, add any missing mipmap levels.*
- `#define OAPISURFACE_NOMIPMAPS 0x0020`  
*Don't create mipmaps. If loaded from file, ignore any mipmap levels present.*
- `#define OAPISURFACE_ALPHA 0x0040`  
*Create an alpha channel for the surface. If loaded from file, add an alpha channel if required.*
- `#define OAPISURFACE_NOALPHA 0x0080`  
*Don't create an alpha channel. If loaded from file, strip any existing alpha channel.*
- `#define OAPISURFACE_UNCOMPRESS 0x0100`  
*Create an uncompressed surface. If loaded from file, uncompress if required.*
- `#define OAPISURFACE_SYSMEM 0x0200`  
*Create the surface in system (host) memory.*

## 15.13 Mesh group editing flags

### 15.13.1 Detailed Description

These constants can be applied to the *flags* field of the `GROUPEDITSPEC` structure to define which parts of a mesh group are to be modified.

**Note:**

The `GRPEDIT_SETUSERFLAG`, `GRPEDIT_ADDUSERFLAG` and `GRPEDIT_DELUSERFLAG` flags are mutually exclusive. Only one can be used at a time.

**See also:**

[GROUPEDITSPEC](#), [oapiEditMeshGroup](#)

**Defines**

- #define `GRPEDIT_SETUSERFLAG` 0x0001  
*replace the group's UsrFlag entry with the value in the [GROUPEDITSPEC](#) structure.*
- #define `GRPEDIT_ADDUSERFLAG` 0x0002  
*Add the UsrFlag value to the group's UsrFlag entry.*
- #define `GRPEDIT_DELUSERFLAG` 0x0004  
*Remove the UsrFlag value from the group's UsrFlag entry.*
- #define `GRPEDIT_VTXCRDX` 0x0008  
*Replace vertex x-coordinates.*
- #define `GRPEDIT_VTXCRDY` 0x0010  
*Replace vertex y-coordinates.*
- #define `GRPEDIT_VTXCRDZ` 0x0020  
*Replace vertex z-coordinates.*
- #define `GRPEDIT_VTXCRD` (`GRPEDIT_VTXCRDX` | `GRPEDIT_VTXCRDY` | `GRPEDIT_VTXCRDZ`)  
*Replace vertex coordinates.*
- #define `GRPEDIT_VTXNMLX` 0x0040  
*Replace vertex x-normals.*
- #define `GRPEDIT_VTXNMLY` 0x0080  
*Replace vertex y-normals.*
- #define `GRPEDIT_VTXNMLZ` 0x0100  
*Replace vertex z-normals.*
- #define `GRPEDIT_VTXNML` (`GRPEDIT_VTXNMLX` | `GRPEDIT_VTXNMLY` | `GRPEDIT_VTXNMLZ`)  
*Replace vertex normals.*
- #define `GRPEDIT_VTXTEXU` 0x0200  
*Replace vertex u-texture coordinates.*
- #define `GRPEDIT_VTXTEXV` 0x0400  
*Replace vertex v-texture coordinates.*

- #define [GRPEDIT\\_VTXTEX](#) (GRPEDIT\_VTXTEXU | GRPEDIT\_VTXTEXV)  
*Replace vertex texture coordinates.*
- #define [GRPEDIT\\_VTX](#) (GRPEDIT\_VTXCRD | GRPEDIT\_VTXNML | GRPEDIT\_VTXTEX)  
*Replace vertices.*

## 15.14 Bitflags for EXHAUSTSPEC flags field.

### 15.14.1 Detailed Description

See also:

[EXHAUSTSPEC](#)

#### Defines

- #define [EXHAUST\\_CONSTANTLEVEL](#) 0x0001  
*exhaust level is constant*
- #define [EXHAUST\\_CONSTANTPOS](#) 0x0002  
*exhaust position is constant*
- #define [EXHAUST\\_CONSTANTDIR](#) 0x0004  
*exhaust direction is constant*

## 15.15 Local lighting interface

### 15.15.1 Detailed Description

The classes in this group define local light sources.

See also:

[VESSEL3::AddPointLight](#), [VESSEL3::AddSpotLight](#)

#### Classes

- class [LightEmitter](#)  
*Base class for defining a light source that can illuminate other objects.*
- class [PointLight](#)  
*Class for isotropic point light source.*
- class [SpotLight](#)  
*Class for directed spot light sources.*

## 15.16 Light beacon shape parameters

### 15.16.1 Detailed Description

See also:

[BEACONLIGHTSPEC](#)

#### Defines

- #define [BEACONSHAPE\\_COMPACT](#) 0  
*compact beacon shape*
- #define [BEACONSHAPE\\_DIFFUSE](#) 1  
*diffuse beacon shape*
- #define [BEACONSHAPE\\_STAR](#) 2  
*star-shaped beacon*

## 15.17 Listentryflag

See also:

[LISTENTRY](#)

## 15.18 Listclbkflag

See also:

[LISTENTRY](#)

## 15.19 Animation flags

### 15.19.1 Detailed Description

See also:

[VESSEL::AddAnimationComponent](#)

#### Defines

- #define [LOCALVERTEXLIST](#) ((UINT)(-1))  
*flags animation component as explicit vertex list*
- #define [MAKEGROUPARRAY](#)(x) ((UINT\*)x)  
*casts a vertex array into a group*

## 15.20 Identifiers for frames of reference

### Enumerations

- enum `REFFRAME` { `FRAME_GLOBAL`, `FRAME_LOCAL`, `FRAME_REFLOCAL`, `FRAME_HORIZON` }

*Identifiers for frames of reference.*

### 15.20.1 Enumeration Type Documentation

#### 15.20.1.1 enum REFFRAME

Identifiers for frames of reference.

##### Enumerator:

`FRAME_GLOBAL` global (ecliptic) frame  
`FRAME_LOCAL` local object frame  
`FRAME_REFLOCAL` local reference object frame  
`FRAME_HORIZON` local horizon frame

## 15.21 Thruster and thruster-group parameters

### Enumerations

- enum `ENGINETYPE` { `ENGINE_MAIN`, `ENGINE_RETRO`, `ENGINE_HOVER`, `ENGINE_ATTITUDE` }

*Thruster group identifiers (obsolete).*

- enum `EXHAUSTTYPE` { `EXHAUST_MAIN`, `EXHAUST_RETRO`, `EXHAUST_HOVER`, `EXHAUST_CUSTOM` }
- enum `THGROUP_TYPE` {  
    `THGROUP_MAIN`, `THGROUP_RETRO`, `THGROUP_HOVER`, `THGROUP_ATT_PITCHUP`,  
    `THGROUP_ATT_PITCHDOWN`, `THGROUP_ATT_YAWLEFT`, `THGROUP_ATT_YAWRIGHT`,  
    `THGROUP_ATT_BANKLEFT`,  
    `THGROUP_ATT_BANKRIGHT`,       `THGROUP_ATT_RIGHT`,       `THGROUP_ATT_LEFT`,  
    `THGROUP_ATT_UP`,  
    `THGROUP_ATT_DOWN`,       `THGROUP_ATT_FORWARD`,       `THGROUP_ATT_BACK`,  
    `THGROUP_USER` = 0x40 }

*Thruster group types.*

### 15.21.1 Enumeration Type Documentation

#### 15.21.1.1 enum ENGINETYPE

Thruster group identifiers (obsolete).

##### Enumerator:

`ENGINE_MAIN` main thrusters

***ENGINE\_RETRO*** retro thrusters  
***ENGINE\_HOVER*** hover thrusters  
***ENGINE\_ATTITUDE*** attitude (RCS) thrusters

### 15.21.1.2 enum THGROUP\_TYPE

Thruster group types.

See also:

[VESSEL::CreateThrusterGroup](#)

Enumerator:

***THGROUP\_MAIN*** main thrusters  
***THGROUP\_RETRO*** retro thrusters  
***THGROUP\_HOVER*** hover thrusters  
***THGROUP\_ATT\_PITCHUP*** rotation: pitch up  
***THGROUP\_ATT\_PITCHDOWN*** rotation: pitch down  
***THGROUP\_ATT\_YAWLEFT*** rotation: yaw left  
***THGROUP\_ATT\_YAWRIGHT*** rotation: yaw right  
***THGROUP\_ATT\_BANKLEFT*** rotation: bank left  
***THGROUP\_ATT\_BANKRIGHT*** rotation: bank right  
***THGROUP\_ATT\_RIGHT*** translation: move right  
***THGROUP\_ATT\_LEFT*** translation: move left  
***THGROUP\_ATT\_UP*** translation: move up  
***THGROUP\_ATT\_DOWN*** translation: move down  
***THGROUP\_ATT\_FORWARD*** translation: move forward  
***THGROUP\_ATT\_BACK*** translation: move back  
***THGROUP\_USER*** user-defined group

## 15.22 Airfoil orientation

### Enumerations

- enum [AIRFOIL\\_ORIENTATION](#) { **LIFT\_VERTICAL**, **LIFT\_HORIZONTAL** }
- Lift vector orientation for airfoils.*

### 15.22.1 Enumeration Type Documentation

#### 15.22.1.1 enum AIRFOIL\_ORIENTATION

Lift vector orientation for airfoils.

Defines the orientation of an airfoil by the direction of the lift vector generated (vertical or horizontal).

See also:

[VESSEL::CreateAirfoil](#), [VESSEL::CreateAirfoil2](#), [VESSEL::CreateAirfoil3](#)

Enumerator:

**LIFT\_VERTICAL** lift direction is vertical (e.g. elevator)

**LIFT\_HORIZONTAL** lift direction is horizontal (e.g. rudder)

## 15.23 Aerodynamic control surface types

### Enumerations

- enum [AIRCTRL\\_TYPE](#) {  
    [AIRCTRL\\_ELEVATOR](#), [AIRCTRL\\_RUDDER](#), [AIRCTRL\\_AILERON](#), [AIRCTRL\\_FLAP](#),  
    [AIRCTRL\\_ELEVATORTRIM](#), [AIRCTRL\\_RUDDERTRIM](#) }

*Control surfaces provide attitude and drag control during atmospheric flight.*

### 15.23.1 Enumeration Type Documentation

#### 15.23.1.1 enum AIRCTRL\_TYPE

Control surfaces provide attitude and drag control during atmospheric flight.

See also:

[VESSEL::CreateControlSurface](#),  
[SEL::CreateControlSurface3](#)

[VESSEL::CreateControlSurface2](#),

VES-

Enumerator:

**AIRCTRL\_ELEVATOR** elevator control (pitch control)  
**AIRCTRL\_RUDDER** rudder control (yaw control)  
**AIRCTRL\_AILERON** aileron control (bank control)  
**AIRCTRL\_FLAP** flaps (lift, drag control)  
**AIRCTRL\_ELEVATORTRIM** elevator trim  
**AIRCTRL\_RUDDERTRIM** rudder trim

## 15.24 Control surface axis orientation

### Defines

- #define [AIRCTRL\\_AXIS\\_AUTO](#) 0

*Constants to define the rotation axis and direction of aerodynamic control surfaces.*

- #define [AIRCTRL\\_AXIS\\_YPOS](#) 1

*y-axis (vertical), positive rotation*

- #define [AIRCTRL\\_AXIS\\_YNEG](#) 2

*y-axis (vertical), negative rotation*

- #define AIRCTRL\_AXIS\_XPOS 3  
*x-axis (transversal), positive rotation*
- #define AIRCTRL\_AXIS\_XNEG 4  
*x-axis (transversal), negative rotation*

#### 15.24.1 Define Documentation

##### 15.24.1.1 #define AIRCTRL\_AXIS\_AUTO 0

Constants to define the rotation axis and direction of aerodynamic control surfaces.

See also:

[VESSEL::CreateControlSurface](#), [VESSEL::CreateControlSurface2](#), [VESSEL::CreateControlSurface3](#) automatic orientation

## 15.25 Identifiers for visual events

### 15.25.1 Detailed Description

\*\*\*\*\*

These constants define events that are sent from the Orbiter core to visual instances in a graphics client. The client receives these notifications via the [oapi::GraphicsClient::clbkVisEvent](#) callback function, where the first parameter is the event identifier, and the second parameter is a message-specific context value.

#### Defines

- #define EVENT\_VESSEL\_INSMESH 0  
*Insert a mesh (context: mesh index).*
- #define EVENT\_VESSEL\_DELMESH 1  
*Delete a mesh (context: mesh index, or -1 for all).*
- #define EVENT\_VESSEL\_MESHVISMODE 2  
*Set mesh visibility mode (context: mesh index).*
- #define EVENT\_VESSEL\_RESETANIM 3  
*Reset animations.*
- #define EVENT\_VESSEL\_CLEARANIM 4  
*Clear all animations (context: UINT (1=reset animations, 0=leave animations at current state)).*
- #define EVENT\_VESSEL\_DELANIM 5  
*Delete an animation (context: animation index).*
- #define EVENT\_VESSEL\_NEWANIM 6  
*Create a new animation (context: animation index).*

- #define EVENT\_VESSEL\_MESHOF 7  
*Shift a mesh (context: mesh index).*
- #define EVENT\_VESSEL\_MODMESHGROUP 8  
*A mesh group has been modified.*

## 15.26 Navigation mode identifiers

### 15.26.1 Detailed Description

These constants are used to refer to the built-in "auto-navigation" modes, mostly for maintaining specific vessel attitudes via use of RCS thrusters.

See also:

[VESSEL::ActivateNavmode](#), [VESSEL::DeactivateNavmode](#), [VESSEL::ToggleNavmode](#), [VESSEL::GetNavmodeState](#)

### Defines

- #define NAVMODE\_KILLROT 1  
*"Kill rotation" mode*
- #define NAVMODE\_HLEVEL 2  
*"Hold level with horizon" mode*
- #define NAVMODE\_PROGRADE 3  
*"Prograde" mode*
- #define NAVMODE\_RETROGRADE 4  
*"Retrograde" mode*
- #define NAVMODE\_NORMAL 5  
*"Normal to orbital plane" mode*
- #define NAVMODE\_ANTINORMAL 6  
*"Anti-normal to orbital plane" mode*
- #define NAVMODE\_HOLDALT 7  
*"Hold altitude" mode*

## 15.27 Manual control mode identifiers

### 15.27.1 Detailed Description

Constants used to identify attitude control modes for manual input.

See also:

[VESSEL::GetManualControlLevel](#)

#### Defines

- #define **MANCTRL\_ATTMODE** 0  
*current attitude mode*
- #define **MANCTRL\_REVMODE** 1  
*reverse of current attitude mode*
- #define **MANCTRL\_ROTMODE** 2  
*rotational attitude modes only*
- #define **MANCTRL\_LINMODE** 3  
*linear attitude modes only*
- #define **MANCTRL\_ANYMODE** 4  
*rotational and linear modes*

## 15.28 Manual control device identifiers

### 15.28.1 Detailed Description

Constants used to identify manual input devices.

See also:

[VESSEL::GetManualControlLevel](#)

#### Defines

- #define **MANCTRL\_KEYBOARD** 0  
*keyboard input*
- #define **MANCTRL\_JOYSTICK** 1  
*joystick input*
- #define **MANCTRL\_ANYDEVICE** 2  
*input from any device*

## 15.29 RCS mode identifiers

### 15.29.1 Detailed Description

These constants are used to define the operation mode of the reaction control system (RCS) of a vessel.

See also:

[VESSEL::GetAttitudeMode](#), [VESSEL::SetAttitudeMode](#)

**Defines**

- #define `RCS_NONE` 0  
*None (RCS off).*
- #define `RCS_ROT` 1  
*Rotational mode.*
- #define `RCS_LIN` 2  
*Linear (translational) mode.*

## 15.30 HUD mode identifiers

### 15.30.1 Detailed Description

These constants are used to refer to the built-in HUD (head-up display) modes.

**Defines**

- #define `HUD_NONE` 0  
*No mode (turn HUD off).*
- #define `HUD_ORBIT` 1  
*Orbit HUD mode.*
- #define `HUD_SURFACE` 2  
*Surface HUD mode.*
- #define `HUD_DOCKING` 3  
*Docking HUD mode.*

## 15.31 MFD mode identifiers

### 15.31.1 Detailed Description

These constants are used to refer to the built-in `MFD` (multifunctional display) modes.

**Defines**

- #define `MFD_REFRESHBUTTONS` -1  
*Refresh `MFD` buttons.*
- #define `MFD_NONE` 0  
*No mode (turn `MFD` off).*
- #define `MFD_ORBIT` 1  
*Orbit `MFD` mode.*

- #define **MFD\_SURFACE** 2  
*Surface MFD mode.*
- #define **MFD\_MAP** 3  
*Map MFD mode.*
- #define **MFD\_HSI** 4  
*HSI (horizontal situation indicator) MFD mode.*
- #define **MFD\_LANDING** 5  
*VTOL support MFD mode.*
- #define **MFD\_DOCKING** 6  
*Docking support MFD mode.*
- #define **MFD\_OPLANEALIGN** 7  
*Orbital plane alignment MFD mode.*
- #define **MFD\_OSYNC** 8  
*Orbit synchronisation MFD mode.*
- #define **MFD\_TRANSFER** 9  
*Transfer orbit MFD mode.*
- #define **MFD\_COMMS** 10  
*Communications MFD mode.*
- #define **MFD\_USERTYPE** 64  
*User-defined MFD mode.*
- #define **BUILTIN\_MFD\_MODES** 10  
*Number of built-in MFD modes.*

## 15.32 MFD identifiers

### Defines

- #define **MAXMFD** 10  
*Max. number of MFD displays per panel.*
- #define **MFD\_LEFT** 0  
*Left default MFD display.*
- #define **MFD\_RIGHT** 1  
*Right default MFD display.*
- #define **MFD\_USER1** 2

*User-defined MFD display 1.*

- #define **MFD\_USER2** 3

*User-defined MFD display 2.*

- #define **MFD\_USER3** 4

*User-defined MFD display 3.*

- #define **MFD\_USER4** 5

*User-defined MFD display 4.*

- #define **MFD\_USER5** 6

*User-defined MFD display 5.*

- #define **MFD\_USER6** 7

*User-defined MFD display 6.*

- #define **MFD\_USER7** 8

*User-defined MFD display 7.*

- #define **MFD\_USER8** 9

*User-defined MFD display 8.*

## 15.33 Panel neighbour identifiers

### 15.33.1 Detailed Description

See also:

[oapiSwitchPanel](#)

#### Defines

- #define **PANEL\_LEFT** 0

*left neighbour*

- #define **PANEL\_RIGHT** 1

*right neighbour*

- #define **PANEL\_UP** 2

*above neighbour*

- #define **PANEL\_DOWN** 3

*below neighbour*

## 15.34 Panel redraw event identifiers

### 15.34.1 Detailed Description

These constants are used to refer to cockpit area redraw event types during panel area registration and by the event handlers.

#### Defines

- #define `PANEL_REDRAW_NEVER` 0x0000  
*Don't generate redraw events.*
- #define `PANEL_REDRAW_ALWAYS` 0x0001  
*Generate event at each frame.*
- #define `PANEL_REDRAW_MOUSE` 0x0002  
*Generate event on mouse event.*
- #define `PANEL_REDRAW_INIT` 0x0003  
*Initialisation event.*
- #define `PANEL_REDRAW_USER` 0x0004  
*User-generated event.*
- #define `PANEL_REDRAW_GDI` 0x1000  
*Allow GDI access during redraw events.*
- #define `PANEL_REDRAW_SKETCHPAD` 0x2000  
*Allow Sketchpad access during redraw events.*

## 15.35 Mouse event identifiers

### 15.35.1 Detailed Description

These constants are used to refer to cockpit mouse event types during panel area registration and by the event handlers.

#### Note:

`PANEL_MOUSE_IGNORE` and `PANEL_MOUSE_ONREPLAY` are used only during area registration. Areas with the `PANEL_MOUSE_IGNORE` attribute never generate mouse events. Areas with the `PANEL_MOUSE_ONREPLAY` attribute generate mouse events also during replay sessions (off by default).

#### Defines

- #define `PANEL_MOUSE_IGNORE` 0x00  
*Don't generate mouse events.*

- #define **PANEL\_MOUSE\_LBUTTONDOWN** 0x01  
*Left button down event.*
- #define **PANEL\_MOUSE\_RBUTTONDOWN** 0x02  
*Right button down event.*
- #define **PANEL\_MOUSE\_LBUTTONUP** 0x04  
*Left button release event.*
- #define **PANEL\_MOUSE\_RBUTTONUP** 0x08  
*Right button release event.*
- #define **PANEL\_MOUSE\_LBRESSED** 0x10  
*Left button down (continuous).*
- #define **PANEL\_MOUSE\_RBRESSED** 0x20  
*Right button down (continuous).*
- #define **PANEL\_MOUSE\_DOWN** 0x03  
*Composite down event.*
- #define **PANEL\_MOUSE\_UP** 0x0C  
*Composite release event.*
- #define **PANEL\_MOUSE\_PRESSED** 0x30  
*Composite down (continuous).*
- #define **PANEL\_MOUSE\_ONREPLAY** 0x40  
*Create mouse events during replay.*

## 15.36 Panel area texture mapping identifiers

### 15.36.1 Detailed Description

The constants are used during panel area instantiations for defining the method used for defining how the panel area texture is presented to the redraw callback function.

#### Note:

**PANEL\_MAP\_NONE** is the most efficient option if the area texture is completely redrawn at each redraw event.

**PANEL\_MAP\_BGONREQUEST** is more efficient than **PANEL\_MAP\_BACKGROUND** if the area texture may not need to be updated at each redraw event.

#### Defines

- #define **PANEL\_MAP\_NONE** 0x00  
*area texture is undefined (i.e. should be completely redrawn)*

- #define **PANEL\_MAP\_BACKGROUND** 0x01  
*area texture contains a copy of the panel background*
- #define **PANEL\_MAP\_CURRENT** 0x02  
*area texture contains a copy of the current panel state*
- #define **PANEL\_MAP\_BGONREQUEST** 0x03  
*area texture is undefined, but panel background can be requested*

## 15.37 Generic vessel message identifiers

### Defines

- #define **VMSG\_LUAINTERPRETER** 0x0001  
*initialise Lua interpreter*
- #define **VMSG\_LUAINSTANCE** 0x0002  
*create Lua vessel instance*
- #define **VMSG\_USER** 0x1000  
*base index for user-defined messages*

## 15.38 Vessel mesh visibility flags

### 15.38.1 Detailed Description

These constants determine the visibility of vessel meshes in specific camera modes.

#### See also:

[VESSEL::SetMeshVisibilityMode](#), [VESSEL::GetMeshVisibilityMode](#)

### Defines

- #define **MESHVIS\_NEVER** 0x00  
*Mesh is never visible.*
- #define **MESHVIS\_EXTERNAL** 0x01  
*Mesh is visible in external views.*
- #define **MESHVIS\_COCKPIT** 0x02  
*Mesh is visible in internal (cockpit) views.*
- #define **MESHVIS\_ALWAYS** (MESHVIS\_EXTERNAL|MESHVIS\_COCKPIT)  
*Mesh is always visible.*
- #define **MESHVIS\_VC** 0x04  
*Mesh is only visible in virtual cockpit internal views.*

- #define **MESHVIS\_EXTPASS** 0x10

*Visibility modifier: render mesh during external pass, even for internal views.*

## 15.39 Navigation radio transmitter types

### 15.39.1 Detailed Description

See also:

[oapiGetNavType](#)

#### Defines

- #define **TRANSMITTER\_NONE** 0
- #define **TRANSMITTER\_VOR** 1
- #define **TRANSMITTER\_VTOL** 2
- #define **TRANSMITTER\_ILS** 3
- #define **TRANSMITTER\_IDS** 4
- #define **TRANSMITTER\_XPDR** 5

## 15.40 Object parameter flags

### 15.40.1 Detailed Description

Used by [oapiGetObjectParam\(\)](#)

#### Defines

- #define **OBJPRM\_PLANET\_SURFACEMAXLEVEL** 0x0001  
*Max. resolution level for planet surface rendering. (Parameter type: DWORD).*
- #define **OBJPRM\_PLANET\_SURFACERIPPLE** 0x0002  
*Flag for ripple effect on reflective surfaces (Parameter type: bool).*
- #define **OBJPRM\_PLANET\_HAZEEXTENT** 0x0003  
*Bleed-in factor of atmospheric haze into planet disc. (Parameter type: double; range: 0-0.9).*
- #define **OBJPRM\_PLANET\_HAZEDENSITY** 0x0004  
*Density at which the horizon haze is rendered (basic density is calculated from atmospheric density) Default: 1.0. (Parameter type: double).*
- #define **OBJPRM\_PLANET\_HAZESHIFT** 0x0005
- #define **OBJPRM\_PLANET\_HAZECOLOUR** 0x0006
- #define **OBJPRM\_PLANET\_FOGPARAM** 0x0007
- #define **OBJPRM\_PLANET\_SHADOWCOLOUR** 0x0008
- #define **OBJPRM\_PLANET\_HASCLOUDS** 0x0009
- #define **OBJPRM\_PLANET\_CLOUDALT** 0x000A
- #define **OBJPRM\_PLANET\_CLOUDROTATION** 0x000B

- #define **OBJPRM\_PLANET\_CLOUDSHADOWCOL** 0x000C
- #define **OBJPRM\_PLANET\_CLOUDMICROTEX** 0x000D
- #define **OBJPRM\_PLANET\_CLOUDMICROALTMIN** 0x000E
- #define **OBJPRM\_PLANET\_CLOUDMICROALTMAX** 0x000F
- #define **OBJPRM\_PLANET\_HASRINGS** 0x0010
- #define **OBJPRM\_PLANET\_RINGMINRAD** 0x0011
- #define **OBJPRM\_PLANET\_RINGMAXRAD** 0x0012
- #define **OBJPRM\_PLANET\_ATTENUATIONALT** 0x0013

*Altitude [m] up to which an atmosphere attenuates light cast from the sun on a spacecraft. (Parameter type: double).*

## 15.41 Orbiter API interface methods

### 15.41.1 Detailed Description

The functions in this section provide a general framework to retrieve and set Orbiter simulation parameters from an addon module. For a linear list of oapi functions, constants and enumerations, see [OrbiterAPI.h](#). For vessel-specific parameters see also the [VESSEL](#) class.

#### Modules

- [Object access functions](#)
- [Vessel creation and destruction](#)
- [Body functions](#)
- [Vessel functions](#)
- [Coordinate transformations](#)
- [Camera functions](#)
- [Functions for planetary bodies](#)
- [Surface base interface](#)
- [Time functions](#)
- [Navigation radio transmitter functions](#)
- [Script interpreter functions](#)
- [Visual and mesh functions](#)
- [HUD, MFD and panel functions](#)
- [Drawing support functions](#)
- [Surface functions](#)
- [Custom MFD mode definition](#)
- [Virtual cockpit functions](#)
- [Customisation - custom menu, dialogs](#)
- [File IO Functions](#)
- [Utility functions](#)
- [User input functions](#)
- [Onscreen annotations](#)
- [Obsolete functions](#)

## Functions

- OAPIFUNC bool `oapiRegisterGraphicsClient` (`oapi::GraphicsClient *gc`)  
*Register graphics client class instance.*
- OAPIFUNC int `oapiGetOrbiterVersion` ()  
*Returns the version number of the Orbiter core system.*
- int `oapiGetModuleVersion` ()  
*Returns the API version number against which the module was linked.*
- OAPIFUNC HINSTANCE `oapiGetOrbiterInstance` ()  
*Returns the instance handle for the running Orbiter application.*
- OAPIFUNC const char \* `oapiGetCmdLine` ()  
*Returns a pointer to the command line with which Orbiter was invoked.*
- OAPIFUNC void `oapiGetViewportSize` (DWORD \*w, DWORD \*h, DWORD \*bpp=0)  
*Returns the dimensions of the render viewport.*
- OAPIFUNC double `oapiGetPanelScale` ()  
*Returns the scaling factor for 2-D instrument panels.*
- OAPIFUNC void `oapiRegisterModule` (`oapi::Module *module`)  
*Register a module interface class instance.*
- OAPIFUNC char \* `oapiDebugString` ()  
*Returns a pointer to a string which will be displayed in the lower left corner of the viewport.*
- OAPIFUNC void `oapiGetBarycentre` (OBJHANDLE hObj, VECTOR3 \*bary)  
*Returns the global position of the barycentre of a complete planetary system or a single planet-moons system.*
- OAPIFUNC SURFHANDLE `oapiRegisterExhaustTexture` (char \*name)  
*Request a custom texture for vessel exhaust rendering.*
- OAPIFUNC SURFHANDLE `oapiRegisterReentryTexture` (char \*name)  
*Request a custom texture for vessel reentry flame rendering.*
- OAPIFUNC SURFHANDLE `oapiRegisterParticleTexture` (char \*name)
- OAPIFUNC void `oapiSetShowGrapplePoints` (bool show)
- OAPIFUNC bool `oapiGetShowGrapplePoints` ()
- OAPIFUNC double `oapiGetInducedDrag` (double cl, double A, double e)  
*Aerodynamics helper function.*
- OAPIFUNC double `oapiGetWaveDrag` (double M, double M1, double M2, double M3, double cmax)  
*Aerodynamics helper function.*

### 15.41.2 Function Documentation

#### 15.41.2.1 OAPIFUNC `char* oapiDebugString ()`

Returns a pointer to a string which will be displayed in the lower left corner of the viewport.

##### Returns:

Pointer to debugging string.

##### Note:

This function should only be used for debugging purposes. Do not use it in published modules!  
The returned pointer refers to a global `char[256]` in the Orbiter core. It is the responsibility of the module to ensure that no overflow occurs.  
If the string is written to more than once per time step (either within a single module or by multiple modules) the last state before rendering will be displayed.  
A typical use would be:

```
sprintf (oapiDebugString(), "my value is %f", myvalue);
```

#### 15.41.2.2 OAPIFUNC `void oapiGetBarycentre (OBJHANDLE hObj, VECTOR3 * bary)`

Returns the global position of the barycentre of a complete planetary system or a single planet-moons system.

##### Parameters:

*hObj* celestial body handle

*bary* pointer to vector receiving barycentre data

##### Note:

The barycentre is the centre of mass of a distribution of objects. In this case, all involved celestial bodies are considered point masses, and the barycentre is defined as

$$\vec{r}_B = \left( \sum_i m_i \right)^{-1} \sum_i m_i \vec{r}_i$$

*hObj* must be the handle of a celestial body.

The summation involves the body itself and all its secondaries, e.g. a planet and its moons.

The barycentre of a star (0th level object) is always the origin (0,0,0).

The barycentre of an object without associated secondaries is identical to its position.

#### 15.41.2.3 OAPIFUNC `const char* oapiGetCmdLine ()`

Returns a pointer to the command line with which Orbiter was invoked.

##### Returns:

Pointer to orbiter command line string.

##### Note:

This method can be used to pass custom parameters to a module directly from the orbiter command line.

**15.41.2.4 OAPIFUNC double oapiGetInducedDrag (double *cl*, double *A*, double *e*)**

Aerodynamics helper function.

This is a helper function which is useful when implementing the callback function calculating the aerodynamics coefficients for an airfoil (see [VESSEL::CreateAirfoil](#)). It computes the lift-induced component  $c_{D,i}$  of the drag coefficient as a function of lift coefficient  $c_L$ , wing aspect ratio  $A$ , and wing efficiency factor  $e$ , as

$$c_{D,i} = \frac{c_L^2}{\pi A e}$$

**Parameters:**

- cl*** lift coefficient
- A*** wing aspect ratio
- e*** wing efficiency factor

**Returns:**

Induced drag coefficient  $c_{D,i}$

**Note:**

The full drag coefficient required by the airfoil callback function consists of several components: profile drag  $c_{D,e}$ , induced drag  $c_{D,i}$  and wave drag  $c_{D,w}$

$$c_D = c_{D,e} + c_{D,i} + c_{D,w}$$

where  $c_{D,e}$  is caused by skin friction and pressure components, and  $c_{D,w}$  is a result of the shock wave and flow separation in transonic and supersonic flight.

The wing aspect ratio is defined as  $b^2/S$ , where  $b$  is the wing span, and  $S$  is the wing area.

The efficiency factor depends on the wing shape. The most efficient wings are elliptical, with  $e = 1$ . For all other shapes,  $e < 1$ .

This function can be interpreted slightly differently by moving the angle of attack-dependency of the profile drag into the induced drag component:

$$c_D = c_{D,0} + c'_{D,i} + c_{D,w}$$

where  $c_{D,0}$  is the zero-lift component of the profile drag, and  $c'_{D,i}$  is a modified induced drag obtained by replacing the shape factor  $e$  with the Oswald efficiency factor. See Programmer's Guide for more details.

**15.41.2.5 int oapiGetModuleVersion ()**

Returns the API version number against which the module was linked.

**Returns:**

module version number

**Note:**

Orbiter version numbers are derived from the build date. The version number is constructed as (year100)\*10000 + month\*100 + day, resulting in a decimal version number of the form YYMMDD

**See also:**

[oapiGetOrbiterVersion](#)

**15.41.2.6 OAPIFUNC HINSTANCE oapiGetOrbiterInstance ()**

Returns the instance handle for the running Orbiter application.

**Returns:**

Orbiter instance handle

**15.41.2.7 OAPIFUNC int oapiGetOrbiterVersion ()**

Returns the version number of the Orbiter core system.

**Returns:**

version number

**Note:**

Orbiter version numbers are derived from the build date. The version number is constructed as (year100)\*10000 + month\*100 + day, resulting in a decimal version number of the form YYMMDD

**See also:**

[oapiGetModuleVersion](#)

**15.41.2.8 OAPIFUNC double oapiGetPanelScale ()**

Returns the scaling factor for 2-D instrument panels.

**Returns:**

Panel scaling factor (>0)

**Note:**

This function returns the user-defined panel scaling factor.

The default scaling factor is 1. Values > 1 cause panels to be expanded, values < 1 cause panels to be shrunk.

**15.41.2.9 OAPIFUNC void oapiGetViewportSize (DWORD \* *w*, DWORD \* *h*, DWORD \* *bpp* = 0)**

Returns the dimensions of the render viewport.

**Parameters:**

*w* pointer to viewport width [pixel]

*h* pointer to viewport height [pixel]

*bpp* pointer to colour depth [bits per pixel]

**Note:**

This function writes the viewport width, height and (optionally) colour depth values into the variables pointed to by the function parameters.

For fullscreen modes, the viewport size corresponds to the fullscreen resolution. For windowed modes, the viewport size corresponds to the client area of the render window.

### 15.41.2.10 OAPIFUNC double oapiGetWaveDrag (double *M*, double *M1*, double *M2*, double *M3*, double *cmax*)

Aerodynamics helper function.

This is a helper function which is useful when implementing the callback function calculating the aerodynamics coefficients for an airfoil (see [VESSEL::CreateAirfoil](#)). It uses a simple model to compute the wave drag component of the drag coefficient,  $c_{D,w}$ . Wave drag significantly affects the vessel drag around Mach 1, and falls off towards lower and higher airspeeds. This function uses the following model:

$$c_{D,w} = \begin{cases} 0 & \text{if } M < M_1 \\ c_m \frac{M - M_1}{M_2 - M_1} & \text{if } M_1 < M < M_2 \\ c_m & \text{if } M_2 < M < M_3 \\ c_m \frac{(M_3^2 - 1)^{1/2}}{(M^2 - 1)^{1/2}} & \text{if } M > M_3 \end{cases}$$

where  $0 < M_1 < M_2 < 1 < M_3$  are characteristic Mach numbers, and  $c_m$  is the maximum wave drag coefficient at transonic speeds.

#### Parameters:

- M* current Mach number
- M1* characteristic Mach number
- M2* characteristic Mach number
- M3* characteristic Mach number
- cmax* maximum wave drag coefficient  $c_m$

#### Returns:

Wave drag coefficient  $c_{D,w}$

#### Note:

The model underlying this function assumes a piecewise linear wave drag profile for  $M < M_3$ , and a decay with  $(M^2 - 1)^{-1/2}$  for  $M > M_3$ . If this profile is not suitable for a given airfoil, the programmer must implement wave drag manually.

#### See also:

[oapiGetInducedDrag](#), [VESSEL::CreateAirfoil](#)

### 15.41.2.11 OAPIFUNC SURFHANDLE oapiRegisterExhaustTexture (char \* *name*)

Request a custom texture for vessel exhaust rendering.

#### Parameters:

- name* exhaust texture file name (without path and extension)

#### Returns:

texture handle

#### Note:

The exhaust texture must be stored in DDS format in Orbiter's default texture directory.

If the texture is not found the function returns NULL.

The texture can be used to define custom textures in [VESSEL::AddExhaust](#).

**See also:**

[oapiRegisterReentryTexture](#), [oapiRegisterParticleTexture](#)

**15.41.2.12 OAPIFUNC bool oapiRegisterGraphicsClient (oapi::GraphicsClient \* *gc*)**

Register graphics client class instance.

Graphics clients plugins should use this function to register the class instance instead of [oapiRegisterModule](#).

**Parameters:**

*gc* pointer to graphics client instance

**Returns:**

true if client was registered successfully

**15.41.2.13 OAPIFUNC void oapiRegisterModule (oapi::Module \* *module*)**

Register a module interface class instance.

Plugin modules that use an interface class instance derived from [oapi::Module](#) must register it with this function during module initialisation (typically in the body of [InitModule](#)).

**Parameters:**

*module* pointer to the interface class instance

**Note:**

The DLL should *not* delete the module instance in [ExitModule](#). Orbiter destroys all registered modules automatically when required.

**15.41.2.14 OAPIFUNC SURFHANDLE oapiRegisterReentryTexture (char \* *name*)**

Request a custom texture for vessel reentry flame rendering.

**Parameters:**

*name* reentry texture file name (without path and extension)

**Returns:**

texture handle

**Note:**

The exhaust texture must be stored in DDS format in Orbiter's default texture directory.  
If the texture is not found the function returns NULL.

The texture can be used to define custom textures in [VESSEL::SetReentryTexture\(\)](#).

**See also:**

[oapiRegisterExhaustTexture](#), [oapiRegisterParticleTexture](#)

## 15.42 Object access functions

### Functions

- OAPIFUNC OBJHANDLE oapiGetObjectByName (char \*name)  
*Returns a handle for a named simulation object.*
- OAPIFUNC OBJHANDLE oapiGetObjectByIndex (int index)  
*Returns a handle for an indexed simulation object.*
- OAPIFUNC DWORD oapiGetObjectCount ()  
*Returns the number of objects currently present in the simulation.*
- OAPIFUNC int oapiGetObjectType (OBJHANDLE hObj)  
*Returns the type of an object identified by its handle.*
- OAPIFUNC const void \* oapiGetObjectParam (OBJHANDLE hObj, DWORD paramtype)  
*Returns an object-specific configuration parameter.*
- OAPIFUNC OBJHANDLE oapiGetVesselByName (char \*name)  
*Returns the handle of a vessel identified by its name.*
- OAPIFUNC OBJHANDLE oapiGetVesselByIndex (int index)  
*Returns the handle of a vessel identified by its reference index.*
- OAPIFUNC DWORD oapiGetVesselCount ()  
*Returns the number of vessels currently present in the simulation.*
- OAPIFUNC bool oapiIsVessel (OBJHANDLE hVessel)  
*Checks if the specified handle is a valid vessel handle.*
- OAPIFUNC OBJHANDLE oapiGetGbodyByName (char \*name)  
*Returns the handle of a celestial body (sun, planet or moon) identified by its name.*
- OAPIFUNC OBJHANDLE oapiGetGbodyByIndex (int index)  
*Returns the handle of a celestial body (sun, planet or moon) indentified by its list index.*
- OAPIFUNC DWORD oapiGetGbodyCount ()  
*Returns the number of celestial bodies (sun, planets and moons) currently present in the simulation.*
- OAPIFUNC OBJHANDLE oapiGetBaseByName (OBJHANDLE hPlanet, char \*name)  
*Returns the handle of a surface base on a given planet or moon.*
- OAPIFUNC OBJHANDLE oapiGetBaseByIndex (OBJHANDLE hPlanet, int index)  
*Returns the handle of a surface base on a planet or moon given by its list index.*
- OAPIFUNC DWORD oapiGetBaseCount (OBJHANDLE hPlanet)  
*Returns the number of surface bases defined for a given planet.*
- OAPIFUNC void oapiGetObjectName (OBJHANDLE hObj, char \*name, int n)

*Returns the name of an object.*

- OAPIFUNC OBJHANDLE oapiGetFocusObject ()  
*Returns the handle for the current focus object.*
- OAPIFUNC OBJHANDLE oapiSetFocusObject (OBJHANDLE hVessel)  
*Switches the input focus to a different vessel object.*
- OAPIFUNC VESSEL \* oapiGetVesselInterface (OBJHANDLE hVessel)  
*Returns a **VESSEL** class instance for a vessel.*
- OAPIFUNC VESSEL \* oapiGetFocusInterface ()  
*Returns the **VESSEL** class instance for the current focus object.*
- OAPIFUNC CELBODY \* oapiGetCelbodyInterface (OBJHANDLE hBody)  
*Returns a **CELBODY** interface instance for a celestial body, if available.*

### 15.42.1 Function Documentation

#### 15.42.1.1 OAPIFUNC OBJHANDLE oapiGetBaseByIndex (OBJHANDLE *hPlanet*, int *index*)

Returns the handle of a surface base on a planet or moon given by its list index.

##### Parameters:

***hPlanet*** handle of the planet or moon on which the base is located  
***index*** list index ( $0 \leq \text{index} < \text{oapiGetBaseCount}(\text{hPlanet})$ )

##### Returns:

Base object handle, or NULL if index out of range.

##### See also:

[oapiGetBaseCount](#), [oapiGetBaseByName](#), [oapiGetBasePlanet](#)

#### 15.42.1.2 OAPIFUNC OBJHANDLE oapiGetBaseByName (OBJHANDLE *hPlanet*, char \* *name*)

Returns the handle of a surface base on a given planet or moon.

##### Parameters:

***hPlanet*** handle of planet or moon on which the base is located  
***name*** base name (not case-sensitive)

##### Returns:

Base object handle, or NULL if base was not found.

##### See also:

[oapiGetBaseByIndex](#), [oapiGetBasePlanet](#)

**15.42.1.3 OAPIFUNC DWORD oapiGetBaseCount (OBJHANDLE *hPlanet*)**

Returns the number of surface bases defined for a given planet.

**Parameters:**

*hPlanet* handle of a planet or moon

**Returns:**

Number of surface bases ( $\geq 0$ ).

**15.42.1.4 OAPIFUNC CELBODY\* oapiGetCelbodyInterface (OBJHANDLE *hBody*)**

Returns a **CELBODY** interface instance for a celestial body, if available.

**Parameters:**

*hBody* handle of a celestial body

**Returns:**

Pointer to the **CELBODY** class instance for the body, or NULL if the body is not controlled by an external module.

**Note:**

*hBody* must be a valid handle for a celestial body (star, planet, moon, etc.), e.g. as obtained from [oapiGetGbodyByName](#). Passing a handle of any other type will result in undefined behaviour.

Only celestial bodies controlled by external plugin modules have access to a **CELBODY** instance. Celestial bodies that are updated internally by Orbiter (e.g. using 2-body orbital elements, or dynamic updates) return NULL here.

**15.42.1.5 OAPIFUNC VESSEL\* oapiGetFocusInterface ()**

Returns the **VESSEL** class instance for the current focus object.

**Returns:**

Pointer to an instance of the **VESSEL** class or a derived class, providing an interface for access to the current focus object.

**15.42.1.6 OAPIFUNC OBJHANDLE oapiGetFocusObject ()**

Returns the handle for the current focus object.

**Returns:**

Focus object handle

**Note:**

The focus object is the user-controlled vessel which receives keyboard and joystick input.

This function returns a valid vessel handle during a simulation session (between [oapi::Module::clbkSimulationStart\(\)](#) and [oapi::Module::clbkSimulationEnd\(\)](#))

**See also:**

[oapiSetFocusObject](#)

**15.42.1.7 OAPIFUNC OBJHANDLE oapiGetGbodyByIndex (int *index*)**

Returns the handle of a celestial body (sun, planet or moon) indentified by its list index.

**Parameters:**

*index* object index ( $0 \leq \text{index} < \text{oapiGetGbodyCount}()$ )

**Returns:**

Object handle, or NUL if index out of range.

**See also:**

[oapiGetGbodyCount](#), [oapiGetGbodyByName](#)

**15.42.1.8 OAPIFUNC OBJHANDLE oapiGetGbodyByName (char \* *name*)**

Returns the handle of a celestial body (sun, planet or moon) identified by its name.

**Parameters:**

*name* celestial object name (not case-sensitive)

**Returns:**

Object handle, or NULL if the object could not be found.

**Note:**

Celestial bodies in orbiter are objects that act as sources for gravitational fields.

**See also:**

[oapiGetGbodyByIndex](#)

**15.42.1.9 OAPIFUNC DWORD oapiGetGbodyCount ()**

Returns the number of celestial bodies (sun, planets and moons) currently present in the simulation.

**Returns:**

Number of objects.

**15.42.1.10 OAPIFUNC OBJHANDLE oapiGetObjectByIndex (int *index*)**

Returns a handle for an indexed simulation object.

**Parameters:**

*index* object index ( $0 \leq \text{index} < \text{oapiGetObjectCount}()$ )

**Returns:**

object handle

**Note:**

Objects can be created and deleted during a simulation session. Therefore the list index of a given object and the range of valid list indices can change.

A typical use for accessing objects by index is in a loop running over all present objects:

```
for (int i = 0; i < oapiGetObjectCount(); i++) {  
    OBJHANDLE hObj = oapiGetObjectByIndex (i);  
    // do something with hObj  
}
```

**See also:**

[oapiGetObjectByName](#), [oapiGetObjectType](#)

**15.42.1.11 OAPIFUNC OBJHANDLE oapiGetObjectByName (char \* *name*)**

Returns a handle for a named simulation object.

**Parameters:**

*name* object name

**Returns:**

object handle

**Note:**

Objects can be vessels, planets, moons or suns.

A return value of NULL indicates that the object was not found.

The name is not case-sensitive ("Jupiter" will also match "jupiter" or "JUPITER").

Surface base handles cannot be retrieved with this method, because a planet handle is required in addition to the base name to uniquely identify the base. Use [oapiGetBaseByName\(\)](#) or [oapiGetBaseByIndex\(\)](#) instead.

**See also:**

[oapiGetObjectByIndex](#), [oapiGetVesselByName](#), [oapiGetGbodyByName](#), [oapiGetBaseByName](#), [oapiGetObjectType](#)

**15.42.1.12 OAPIFUNC DWORD oapiGetObjectCount ()**

Returns the number of objects currently present in the simulation.

**Returns:**

object count

**See also:**

[oapiGetObjectByName](#), [oapiGetObjectType](#)

**15.42.1.13 OAPIFUNC void oapiGetObjectName (OBJHANDLE *hObj*, char \* *name*, int *n*)**

Returns the name of an object.

**Parameters:**

*hObj* object handle  
*name* pointer to character array to receive object name  
*n* length of character array

**Note:**

*name* must be allocated to at least size *n* by the calling function.  
If the string buffer is not long enough to hold the object name, the name is truncated.

**15.42.1.14 OAPIFUNC const void\* oapiGetObjectParam (OBJHANDLE *hObj*, DWORD *param-type*)**

Returns an object-specific configuration parameter.

**Parameters:**

*hObj* object handle  
*param-type* parameter identifier (see [Object parameter flags](#))

**Returns:**

pointer to parameter value

**Note:**

This function returns the current value of a configuration parameter for a given object (e.g. planet).  
The type of the return value depends on the parameter. The generic void pointer must be cast into the appropriate parameter type. Example:

```
bool *bClouds = (bool*)oapiGetObjectParam (hObj, OBJPRM_PLANET_HASCLOUDS);
```

**See also:**

[Object parameter flags](#)

**15.42.1.15 OAPIFUNC int oapiGetObjectType (OBJHANDLE *hObj*)**

Returns the type of an object identified by its handle.

**Parameters:**

*hObj* object handle

**Returns:**

Integer code identifying the vessel type.

**Note:**

The following type identifiers are currently supported:

OBJTP_INVALID	invalid object handle
OBJTP_GENERIC	generic object (not currently used)
OBJTP_CBODY	generic celestial body (not currently used)
OBJTP_STAR	star
OBJTP_PLANET	planet (used for all celestial bodies that are not stars, including moons, comets, etc.)
OBJTP_VESSEL	vessel (spacecraft, space stations, etc.)
OBJTP_SURFBASE	surface base (spaceport)

**See also:**

[oapiGetObjectParam](#), [oapiGetObjectCount](#)

**15.42.1.16 OAPIFUNC OBJHANDLE oapiGetVesselByIndex (int *index*)**

Returns the handle of a vessel identified by its reference index.

**Parameters:**

*index* object index ( $0 \leq \text{index} < \text{oapiGetVesselCount}()$ )

**Returns:**

Vessel object handle, or NULL if index out of range.

**Note:**

The index of a vessel can change during the simulation if vessels are created or destroyed. A typical use for [oapiGetVesselByIndex\(\)](#) would be to implement a loop over all vessels:

```
for (i = 0; i < oapiGetVesselCount(); i++) {
    OBJHANDLE hVessel = oapiGetVesselByIndex (i);
    // do something with hVessel
}
```

**See also:**

[oapiGetVesselByName](#), [oapiGetVesselCount](#)

**15.42.1.17 OAPIFUNC OBJHANDLE oapiGetVesselByName (char \* *name*)**

Returns the handle of a vessel identified by its name.

**Parameters:**

*name* vessel name (not case-sensitive)

**Returns:**

Vessel object handle, or NULL if the vessel could not be found.

**See also:**

[oapiGetVesselByIndex](#)

**15.42.1.18 OAPIFUNC DWORD oapiGetVesselCount ()**

Returns the number of vessels currently present in the simulation.

**Returns:**

Vessel count.

**See also:**

[oapiGetVesselByIndex](#)

**15.42.1.19 OAPIFUNC VESSEL\* oapiGetVesselInterface (OBJHANDLE *hVessel*)**

Returns a [VESSEL](#) class instance for a vessel.

**Parameters:**

*hVessel* vessel handle

**Returns:**

Pointer to an instance of the [VESSEL](#) class or a derived class, providing an interface for access to the specified vessel.

**15.42.1.20 OAPIFUNC bool oapiIsVessel (OBJHANDLE *hVessel*)**

Checks if the specified handle is a valid vessel handle.

**Parameters:**

*hVessel* handle to be tested

**Returns:**

*true* if *hVessel* is a valid vessel handle, *false* otherwise.

**Note:**

This function can be used to test if a previously obtained vessel handle is still valid. A handle becomes invalid if the associated vessel is deleted.

An alternative to using [oapiIsVessel\(\)](#) is monitoring vessel deletions by implementing the [oapi::Module::clbkDeleteVessel\(\)](#) callback function of the module instance.

**See also:**

[oapiGetObjectType](#)

**15.42.1.21 OAPIFUNC OBJHANDLE oapiSetFocusObject (OBJHANDLE *hVessel*)**

Switches the input focus to a different vessel object.

**Parameters:**

*hVessel* handle of vessel to receive input focus

**Returns:**

Handle of vessel losing focus, or NULL if focus did not change.

**Note:**

*hVessel* must refer to a vessel object. Trying to set the focus to a different object type will fail.

**See also:**

[oapiGetFocusObject](#)

## 15.43 Vessel creation and destruction

**Functions**

- OAPIFUNC **OBJHANDLE oapiCreateVessel** (const char \*name, const char \*classname, const **VESSELSTATUS** &status)  
*Creates a new vessel.*
- OAPIFUNC **OBJHANDLE oapiCreateVesselEx** (const char \*name, const char \*classname, const void \*status)  
*Creates a new vessel via a VESSELSTATUSx (x >= 2) interface.*
- OAPIFUNC bool **oapiDeleteVessel** (**OBJHANDLE** hVessel, **OBJHANDLE** hAlternativeCameraTarget=0)  
*Deletes an existing vessel.*

### 15.43.1 Function Documentation

#### 15.43.1.1 OAPIFUNC **OBJHANDLE oapiCreateVessel (const char \* name, const char \* classname, const VESSELSTATUS & status)**

Creates a new vessel.

**Parameters:**

**name** vessel name  
**classname** vessel class name  
**status** initial vessel status

**Returns:**

Handle of the new vessel.

**Note:**

A configuration file for the specified vessel class must exist in the Config or Config/Vessels subdirectory.

[oapiCreateVesselEx](#) is an extended version of this function operating on a more versatile status structure.

**See also:**

[oapiCreateVesselEx](#), **VESSELSTATUS**

**15.43.1.2 OAPIFUNC OBJHANDLE oapiCreateVesselEx (const char \* *name*, const char \* *classname*, const void \* *status*)**

Creates a new vessel via a VESSELSTATUSx ( $x \geq 2$ ) interface.

**Parameters:**

*name* vessel name

*classname* vessel class name

*status* pointer to a VESSELSTATUSx structure

**Returns:**

Handle of the new vessel.

**Note:**

A configuration file for the specified vessel class must exist in the Config or the Config\ Vessels folder, or a subfolder. If the config file is located in a subfolder, the relative path must be included in the *classname* parameter.

*status* must point to a VESSELSTATUSx structure. Currently only [VESSELSTATUS2](#) is supported, but future Orbiter versions may add new interfaces.

During the vessel creation process Orbiter will call the module's [VESSEL2::clbkSetStateEx](#) callback function if it exists.

**See also:**

[oapiCreateVessel](#), [VESSEL2::clbkSetStateEx](#), [VESSELSTATUS2](#)

**15.43.1.3 OAPIFUNC bool oapiDeleteVessel (OBJHANDLE *hVessel*, OBJHANDLE *hAlternativeCameraTarget* = 0)**

Deletes an existing vessel.

**Parameters:**

*hVessel* vessel handle

*hAlternativeCameraTarget* optional new camera target

**Returns:**

*true* if vessel could be deleted.

**Note:**

If the current focus vessel is deleted, Orbiter will switch focus to the closest focus-enabled vessel. If the last focus-enabled vessel is deleted, Orbiter returns to the launchpad.

If the current camera target is deleted, a new camera target can be provided in *hAlternativeCameraTarget*. If not specified, the focus object is used as default camera target.

The actual vessel destruction does not occur until the end of the current frame. Self-destruct calls are therefore permitted.

A vessel will undock all its docking ports before being destructed.

**See also:**

[oapiCreateVessel](#), [oapiCreateVesselEx](#)

## 15.44 Body functions

### Functions

- OAPIFUNC double [oapiGetSize \(OBJHANDLE hObj\)](#)  
*Returns the size (mean radius) of an object.*
- OAPIFUNC double [oapiGetMass \(OBJHANDLE hObj\)](#)  
*Returns the mass of an object. For vessels, this is the total mass, including current fuel mass.*
- OAPIFUNC void [oapiGetGlobalPos \(OBJHANDLE hObj, VECTOR3 \\*pos\)](#)  
*Returns the position of an object in the global reference frame.*
- OAPIFUNC void [oapiGetGlobalVel \(OBJHANDLE hObj, VECTOR3 \\*vel\)](#)  
*Returns the velocity of an object in the global reference frame.*
- OAPIFUNC void [oapiGetRelativePos \(OBJHANDLE hObj, OBJHANDLE hRef, VECTOR3 \\*pos\)](#)  
*Returns the distance vector from hRef to hObj in the ecliptic reference frame.*
- OAPIFUNC void [oapiGetRelativeVel \(OBJHANDLE hObj, OBJHANDLE hRef, VECTOR3 \\*vel\)](#)  
*Returns the velocity difference vector of hObj relative to hRef in the ecliptic reference frame.*

### 15.44.1 Function Documentation

#### 15.44.1.1 OAPIFUNC void oapiGetGlobalPos (OBJHANDLE *hObj*, VECTOR3 \* *pos*)

Returns the position of an object in the global reference frame.

##### Parameters:

***hObj*** object handle  
***pos*** pointer to vector receiving coordinates

##### Note:

The global reference frame is the heliocentric ecliptic system at ecliptic and equinox of J2000.  
Units are meters.

##### See also:

[oapiGetBarycentre](#), [oapiGetGlobalVel](#)

#### 15.44.1.2 OAPIFUNC void oapiGetGlobalVel (OBJHANDLE *hObj*, VECTOR3 \* *vel*)

Returns the velocity of an object in the global reference frame.

##### Parameters:

***hObj*** object handle  
***vel*** pointer to vector receiving velocity data

**Note:**

The global reference frame is the heliocentric ecliptic system at ecliptic and equinox of J2000.  
Units are meters/second.

**See also:**

[oapiGetBarycentre](#), [oapiGetGlobalPos](#)

**15.44.1.3 OAPIFUNC double oapiGetMass (OBJHANDLE *hObj*)**

Returns the mass of an object. For vessels, this is the total mass, including current fuel mass.

**Parameters:**

*hObj* object handle

**Returns:**

object mass [kg]

**See also:**

[oapiGetMaxFuelMass](#), [oapiGetEmptyMass](#)

**15.44.1.4 OAPIFUNC void oapiGetRelativePos (OBJHANDLE *hObj*, OBJHANDLE *hRef*, VECTOR3 \* *pos*)**

Returns the distance vector from hRef to hObj in the ecliptic reference frame.

**Parameters:**

*hObj* object handle

*hRef* reference object handle

*pos* pointer to vector receiving distance data

**Note:**

Results are w.r.t. ecliptic frame at equinox and ecliptic of J2000.0.

**See also:**

[oapiGetBarycentre](#), [oapiGetRelativeVel](#)

**15.44.1.5 OAPIFUNC void oapiGetRelativeVel (OBJHANDLE *hObj*, OBJHANDLE *hRef*, VECTOR3 \* *vel*)**

Returns the velocity difference vector of hObj relative to hRef in the ecliptic reference frame.

**Parameters:**

*hObj* object handle

*hRef* reference object handle

*vel* pointer to vector receiving velocity difference data

**Note:**

Results are w.r.t. ecliptic frame at equinox and ecliptic of J2000.0.

**See also:**

[oapiGetBarycentre](#), [oapiGetRelativePos](#)

**15.44.1.6 OAPIFUNC double oapiGetSize (OBJHANDLE *hObj*)**

Returns the size (mean radius) of an object.

**Parameters:**

*hObj* object handle

**Returns:**

Object size (mean radius) in meter.

**15.45 Vessel functions****Functions**

- OAPIFUNC double [oapiGetEmptyMass \(OBJHANDLE hVessel\)](#)  
*Returns empty mass of a vessel, excluding fuel.*
- OAPIFUNC void [oapiSetEmptyMass \(OBJHANDLE hVessel, double mass\)](#)  
*Set the empty mass of a vessel (excluding fuel).*
- OAPIFUNC double [oapiGetFuelMass \(OBJHANDLE hVessel\)](#)  
*Returns current fuel mass of the first propellant resource of a vessel.*
- OAPIFUNC double [oapiGetMaxFuelMass \(OBJHANDLE hVessel\)](#)  
*Returns maximum fuel capacity of the first propellant resource of a vessel.*
- OAPIFUNC PROPELLANT\_HANDLE [oapiGetPropellantHandle \(OBJHANDLE hVessel, DWORD idx\)](#)  
*Returns an identifier of a vessel's propellant resource.*
- OAPIFUNC double [oapiGetPropellantMass \(PROPELLANT\\_HANDLE ph\)](#)  
*Returns the current fuel mass [kg] of a propellant resource.*
- OAPIFUNC double [oapiGetPropellantMaxMass \(PROPELLANT\\_HANDLE ph\)](#)  
*Returns the maximum capacity [kg] of a propellant resource.*
- OAPIFUNC DOCKHANDLE [oapiGetDockHandle \(OBJHANDLE hVessel, UINT n\)](#)  
*Returns a handle to a vessel docking port.*
- OAPIFUNC OBJHANDLE [oapiGetDockStatus \(DOCKHANDLE dock\)](#)  
*Returns the handle of a vessel docked at a port.*

- OAPIFUNC void `oapiGetFocusGlobalPos (VECTOR3 *pos)`  
*Returns the position of the current focus object in the global reference frame.*
- OAPIFUNC void `oapiGetFocusGlobalVel (VECTOR3 *vel)`  
*Returns the velocity of the current focus object in the global reference frame.*
- OAPIFUNC void `oapiGetFocusRelativePos (OBJHANDLE hRef, VECTOR3 *pos)`  
*Returns the distance vector from hRef to the current focus object.*
- OAPIFUNC void `oapiGetFocusRelativeVel (OBJHANDLE hRef, VECTOR3 *vel)`  
*Returns the velocity difference vector of the current focus object relative to hRef.*
- OAPIFUNC BOOL `oapiGetAltitude (OBJHANDLE hVessel, double *alt)`  
*Returns the altitude of a vessel over a planetary surface.*
- OAPIFUNC BOOL `oapiGetPitch (OBJHANDLE hVessel, double *pitch)`  
*Returns a vessel's pitch angle w.r.t. the local horizon.*
- OAPIFUNC BOOL `oapiGetBank (OBJHANDLE hVessel, double *bank)`  
*Returns a vessel's bank angle w.r.t. the local horizon.*
- OAPIFUNC BOOL `oapiGetHeading (OBJHANDLE hVessel, double *heading)`  
*Returns a vessel's heading (against geometric north) calculated for the local horizon plane.*
- OAPIFUNC BOOL `oapiGetFocusAltitude (double *alt)`  
*Returns the altitude of the current focus vessel over a planetary surface.*
- OAPIFUNC BOOL `oapiGetFocusPitch (double *pitch)`  
*Returns the pitch angle of the current focus vessel w.r.t. the local horizon.*
- OAPIFUNC BOOL `oapiGetFocusBank (double *bank)`  
*Returns the bank angle of the current focus vessel w.r.t. the local horizon.*
- OAPIFUNC BOOL `oapiGetFocusHeading (double *heading)`  
*Returns the heading (against geometric north) of the current focus vessel calculated for the local horizon plane.*
- OAPIFUNC BOOL `oapiGetGroundspeed (OBJHANDLE hVessel, double *groundspeed)`  
*Returns a vessel's ground speed w.r.t. the closest planet or moon.*
- OAPIFUNC bool `oapiGetGroundspeedVector (OBJHANDLE hVessel, REFFRAME frame, VECTOR3 *vel)`  
*Returns a vessel's groundspeed vector w.r.t. the closest planet or moon in the requested frame of reference.*
- OAPIFUNC BOOL `oapiGetAirspeed (OBJHANDLE hVessel, double *airspeed)`  
*Returns a vessel's true airspeed w.r.t. the closest planet or moon.*
- OAPIFUNC bool `oapiGetAirspeedVector (OBJHANDLE hVessel, REFFRAME frame, VECTOR3 *v)`  
*Returns a vessel's true airspeed vector w.r.t. the closest planet or moon in the requested frame of reference.*

- OAPIFUNC BOOL [oapiGetEquPos](#) (**OBJHANDLE** hVessel, double \*longitude, double \*latitude, double \*radius)  
*Returns a vessel's spherical equatorial coordinates (longitude, latitude and radius) with respect to the closest planet or moon.*
- OAPIFUNC BOOL [oapiGetFocusEquPos](#) (double \*longitude, double \*latitude, double \*radius)  
*Returns the current focus vessel's spherical equatorial coordinates (longitude, latitude and radius) with respect to the closest planet or moon.*
- OAPIFUNC void [oapiGetAtm](#) (**OBJHANDLE** hVessel, **ATMPARAM** \*prm, **OBJHANDLE** \*hAtmRef=0)  
*Returns the atmospheric parameters at the current vessel position.*
- OAPIFUNC void [oapiGetEngineStatus](#) (**OBJHANDLE** hVessel, **ENGINESTATUS** \*es)  
*Retrieve the status of main, retro and hover thrusters for a vessel.*
- OAPIFUNC void [oapiGetFocusEngineStatus](#) (**ENGINESTATUS** \*es)  
*Retrieve the engine status for the focus vessel.*
- OAPIFUNC void [oapiSetEngineLevel](#) (**OBJHANDLE** hVessel, **ENGINETYPE** engine, double level)  
*Engage the specified engines.*
- OAPIFUNC int [oapiGetAttitudeMode](#) (**OBJHANDLE** hVessel)  
*Returns a vessel's current attitude thruster mode.*
- OAPIFUNC int [oapiToggleAttitudeMode](#) (**OBJHANDLE** hVessel)  
*Flip a vessel's attitude thruster mode between rotational and linear.*
- OAPIFUNC bool [oapiSetAttitudeMode](#) (**OBJHANDLE** hVessel, int mode)  
*Set a vessel's attitude thruster mode.*
- OAPIFUNC int [oapiGetFocusAttitudeMode](#) ()  
*Returns the current focus vessel's attitude thruster mode (rotational or linear).*
- OAPIFUNC int [oapiToggleFocusAttitudeMode](#) ()  
*Flip the current focus vessel's attitude thruster mode between rotational and linear.*
- OAPIFUNC bool [oapiSetFocusAttitudeMode](#) (int mode)  
*Set the current focus vessel's attitude thruster mode.*

### 15.45.1 Function Documentation

#### 15.45.1.1 OAPIFUNC BOOL [oapiGetAirspeed](#) (**OBJHANDLE** hVessel, double \*airspeed)

Returns a vessel's true airspeed w.r.t. the closest planet or moon.

##### Parameters:

**hVessel** vessel handle, or NULL for focus vessel

*airspeed* pointer to variable receiving airspeed value [m/s]

**Returns:**

Error flag (*false* on error)

**Note:**

This function works even for planets or moons without atmosphere. In that case it returns the ground speed.

**See also:**

[oapiGetAirspeedVector](#), [oapiGetGroundspeed](#), [oapiGetGroundspeedVector](#), [VESSEL::GetAirspeed](#)

### 15.45.1.2 OAPIFUNC bool oapiGetAirspeedVector (OBJHANDLE *hVessel*, REFFRAME *frame*, VECTOR3 \* *v*)

Returns a vessel's true airspeed vector w.r.t. the closest planet or moon in the requested frame of reference.

**Parameters:**

- ← *hVessel* vessel handle, or NULL for focus vessel
- ← *frame* frame of reference flag
- *v* pointer to variable receiving airspeed vector [m/s in x,y,z]

**Returns:**

Error flag (*false* indicates error)

**Note:**

This method returns the true airspeed vector in the requested frame of reference. The airspeed vector is defined as the vessel's velocity vector with respect to the surrounding freestream air flow.

If the vessel is not within an a planetary atmosphere, the returned vector is equal to the groundspeed vector.

Valid entries for *frame* are

- FRAME\_GLOBAL: Returns velocity vector in the global frame of reference
- FRAME\_LOCAL: Returns velocity vector in the vessel's local frame of reference
- FRAME\_REFLOCAL: Returns velocity vector in the celestial reference body's local frame of reference
- FRAME\_HORIZON: Returns velocity vector in the local horizon frame (x = longitudinal component, y = vertical component, z = latitudinal component)

**See also:**

[oapiGetAirspeed](#), [oapiGetGroundspeedVector](#), [oapiGetGroundspeed](#), [VESSEL::GetAirspeedVector](#)

### 15.45.1.3 OAPIFUNC BOOL oapiGetAltitude (OBJHANDLE *hVessel*, double \* *alt*)

Returns the altitude of a vessel over a planetary surface.

**Parameters:**

*hVessel* vessel handle

*alt* pointer to variable receiving altitude value

**Returns:**

Error flag (*false* on failure)

**Note:**

Unit is meter [m]

Returns altitude above closest planet.

Altitude is measured above mean planet radius (as defined by SIZE parameter in planet's cfg file)

The handle passed to the function must refer to a vessel.

**15.45.1.4 OAPIFUNC void oapiGetAtm (OBJHANDLE *hVessel*, ATMPARAM \**prm*, OBJHANDLE \**hAtmRef* = 0)**

Returns the atmospheric parameters at the current vessel position.

**Parameters:**

← *hVessel* vessel handle

→ *prm* pointer to **ATMPARAM** structure receiving atmospheric parameters.

→ *hAtmRef* pointer to handle receiving the atmosphere reference body.

**Note:**

If *hVessel* == NULL, the current focus vessel is used for the calculation.

If *hAtmRef* != NULL, it receives the handle of the celestial body contributing the atmospheric parameters.

If the vessel is not within range of any planet atmosphere model, all fields of the *prm* structure are set to 0. If applicable, *\*hAtmRef* is set to NULL.

Currently, atmospheric values only depend on altitude, and don't take into account local weather variations.

**15.45.1.5 OAPIFUNC int oapiGetAttitudeMode (OBJHANDLE *hVessel*)**

Returns a vessel's current attitude thruster mode.

**Parameters:**

*hVessel* vessel handle

**Returns:**

Current attitude mode (0=disabled or not available, 1=rotational, 2=linear)

**Note:**

The handle must refer to a vessel. This function does not support other object types.

**See also:**

[oapiToggleAttitudeMode](#), [oapiSetAttitudeMode](#)

**15.45.1.6 OAPIFUNC BOOL oapiGetBank (OBJHANDLE *hVessel*, double \* *bank*)**

Returns a vessel's bank angle w.r.t. the local horizon.

**Parameters:**

*hVessel* vessel handle  
*bank* pointer to variable receiving bank value

**Returns:**

Error flag (*false* on failure)

**Note:**

Unit is radian [rad]  
Returns bank angle w.r.t. closest planet  
The local horizon is the plane whose normal is defined by the distance vector from the planet centre to the vessel.  
The handle passed to the function must refer to a vessel.

**See also:**

[oapiGetHeading](#), [oapiGetPitch](#), [oapiGetAltitude](#)

**15.45.1.7 OAPIFUNC DOCKHANDLE oapiGetDockHandle (OBJHANDLE *hVessel*, UINT *n*)**

Returns a handle to a vessel docking port.

**Parameters:**

*hVessel* vessel handle  
*n* docking port index ( $\geq 0$ )

**Returns:**

docking port handle, or NULL if index is out of range

**See also:**

[VESSEL::GetDockHandle](#)

**15.45.1.8 OAPIFUNC OBJHANDLE oapiGetDockStatus (DOCKHANDLE *dock*)**

Returns the handle of a vessel docked at a port.

**Parameters:**

*dock* docking port handle

**Returns:**

Handle of docked vessel, or NULL if no vessel is docked at the port.

**See also:**

[oapiGetDockHandle](#), [VESSEL::GetDockStatus](#)

**15.45.1.9 OAPIFUNC double oapiGetEmptyMass (OBJHANDLE *hVessel*)**

Returns empty mass of a vessel, excluding fuel.

**Parameters:**

*hVessel* vessel handle

**Returns:**

empty vessel mass [kg]

**Note:**

*hVessel* must be a vessel handle. Other object types are invalid.

Do not rely on a constant empty mass. Structural changes (e.g. discarding a rocket stage) will affect the empty mass.

For multistage configurations, the fuel mass of all currently inactive stages contributes to the empty mass. Only the fuel mass of active stages is excluded.

**15.45.1.10 OAPIFUNC void oapiGetEngineStatus (OBJHANDLE *hVessel*, ENGINESTATUS \* *es*)**

Retrieve the status of main, retro and hover thrusters for a vessel.

**Parameters:**

*hVessel* vessel handle

*es* pointer to an [ENGINESTATUS](#) structure which will receive the engine level parameters

**Note:**

The main/retro engine level has a range of [-1,+1]. A positive value indicates engaged main/disengaged retro thrusters, a negative value indicates engaged retro/disengaged main thrusters. Main and retro thrusters cannot be engaged simultaneously. For vessels without retro thrusters the valid range is [0,+1]. The valid range for hover thrusters is [0,+1].

[ENGINESTATUS](#) has the following components:

```
typedef struct {
    double main;      // -1 (full retro) .. +1 (full main)
    double hover;    // 0 .. +1 (full hover)
    int attmode;     // 0=rotation, 1=translation
} ENGINESTATUS;
```

**15.45.1.11 OAPIFUNC BOOL oapiGetEquPos (OBJHANDLE *hVessel*, double \* *longitude*, double \* *latitude*, double \* *radius*)**

Returns a vessel's spherical equatorial coordinates (longitude, latitude and radius) with respect to the closest planet or moon.

**Parameters:**

*hVessel* vessel handle

*longitude* pointer to variable receiving longitude value [rad]

*latitude* pointer to variable receiving latitude value [rad]

*radius* pointer to variable receiving radius value [m]

**Returns:**

Error flag (*false* on failure)

**Note:**

The handle passed to the function must refer to a vessel.

#### 15.45.1.12 OAPIFUNC BOOL oapiGetFocusAltitude (double \* *alt*)

Returns the altitude of the current focus vessel over a planetary surface.

**Parameters:**

*alt* pointer to variable receiving altitude value [m]

**Returns:**

Error flag (*false* on failure )

#### 15.45.1.13 OAPIFUNC int oapiGetFocusAttitudeMode ()

Returns the current focus vessel's attitude thruster mode (rotational or linear).

**Returns:**

Current attitude mode (0=disabled or not available, 1=rotational, 2=linear)

#### 15.45.1.14 OAPIFUNC BOOL oapiGetFocusBank (double \* *bank*)

Returns the bank angle of the current focus vessel w.r.t. the local horizon.

**Parameters:**

*bank* pointer to variable receiving bank angle [rad]

**Returns:**

Error flag (*false* on failure)

**See also:**

[oapiGetFocusHeading](#), [oapiGetFocusPitch](#), [oapiGetFocusAltitude](#)

#### 15.45.1.15 OAPIFUNC void oapiGetFocusEngineStatus (ENGINESTATUS \* *es*)

Retrieve the engine status for the focus vessel.

**Parameters:**

*es* pointer to an [ENGINESTATUS](#) structure which will receive the engine level parameters.

**See also:**

[oapiGetEngineStatus](#)

**15.45.1.16 OAPIFUNC BOOL oapiGetFocusEquPos (double \*longitude, double \*latitude, double \*radius)**

Returns the current focus vessel's spherical equatorial coordinates (longitude, latitude and radius) with respect to the closest planet or moon.

**Parameters:**

*longitude* pointer to variable receiving longitude value [rad]

*latitude* pointer to variable receiving latitude value [rad]

*radius* pointer to variable receiving radius value [m]

**Returns:**

Error flag (*false* on failure)

**15.45.1.17 OAPIFUNC void oapiGetFocusGlobalPos (VECTOR3 \*pos)**

Returns the position of the current focus object in the global reference frame.

**Parameters:**

*pos* pointer to vector receiving coordinates

**Note:**

The global reference frame is the heliocentric ecliptic system at ecliptic and equinox of J2000.0.  
Units are meters.

**See also:**

[oapiGetFocusGlobalVel](#)

**15.45.1.18 OAPIFUNC void oapiGetFocusGlobalVel (VECTOR3 \*vel)**

Returns the velocity of the current focus object in the global reference frame.

**Parameters:**

*vel* pointer to vector receiving velocity data

**Note:**

The global reference frame is the heliocentric ecliptic system at ecliptic and equinox of J2000.  
Units are meters/second.

**See also:**

[oapiGetFocusGlobalPos](#)

**15.45.1.19 OAPIFUNC BOOL oapiGetFocusHeading (double \* *heading*)**

Returns the heading (against geometric north) of the current focus vessel calculated for the local horizon plane.

**Parameters:**

*heading* pointer to variable receiving heading value [rad]

**Returns:**

Error flag (*false* on failure)

**See also:**

[oapiGetFocusBank](#), [oapiGetFocusPitch](#), [oapiGetFocusAltitude](#)

**15.45.1.20 OAPIFUNC BOOL oapiGetFocusPitch (double \* *pitch*)**

Returns the pitch angle of the current focus vessel w.r.t. the local horizon.

**Parameters:**

*pitch* pointer to variable receiving pitch value

**Returns:**

Error flag (*false* on failure)

**See also:**

[oapiGetFocusBank](#), [oapiGetFocusHeading](#), [oapiGetFocusAltitude](#)

**15.45.1.21 OAPIFUNC void oapiGetFocusRelativePos (OBJHANDLE *hRef*, VECTOR3 \* *pos*)**

Returns the distance vector from hRef to the current focus object.

**Parameters:**

*hRef* reference object handle

*pos* pointer to vector receiving distance data

**Note:**

Results are w.r.t. ecliptic frame at equinox and ecliptic of J2000.0.

**See also:**

[oapiGetFocusRelativeVel](#)

**15.45.1.22 OAPIFUNC void oapiGetFocusRelativeVel (OBJHANDLE *hRef*, VECTOR3 \* *vel*)**

Returns the velocity difference vector of the current focus object relative to hRef.

**Parameters:**

*hRef* reference object handle  
*vel* pointer to vector receiving velocity difference data

**Note:**

Results are w.r.t. ecliptic frame at equinox and ecliptic of J2000.0.

**See also:**

[oapiGetFocusRelativePos](#)

**15.45.1.23 OAPIFUNC double oapiGetFuelMass (OBJHANDLE *hVessel*)**

Returns current fuel mass of the first propellant resource of a vessel.

**Parameters:**

*hVessel* vessel handle

**Returns:**

Current fuel mass [kg]

**Note:**

This function is equivalent to

```
oapiGetPropellantMass (oapiGetPropellantHandle (hVessel, 0))
```

*hVessel* must be a vessel handle. Other object types are invalid.

For multistage configurations, this returns the current fuel mass of active stages only.

**See also:**

[oapiGetMaxFuelMass](#), [oapiGetEmptyMass](#)

**15.45.1.24 OAPIFUNC BOOL oapiGetGroundspeed (OBJHANDLE *hVessel*, double \* *ground-speed*)**

Returns a vessel's ground speed w.r.t. the closest planet or moon.

**Parameters:**

*hVessel* vessel handle, or NULL for focus vessel  
*ground-speed* pointer to variable receiving ground speed value [m/s]

**Returns:**

Error flag (*false* on error)

**See also:**

[oapiGetGroundspeedVector](#), [oapiGetAirspeed](#), [oapiGetAirspeedVector](#)

**15.45.1.25 OAPIFUNC bool oapiGetGroundspeedVector (OBJHANDLE *hVessel*, REFFRAME *frame*, VECTOR3 \* *vel*)**

Returns a vessel's groundspeed vector w.r.t. the closest planet or moon in the requested frame of reference.

**Parameters:**

- ← *hVessel* vessel handle, or NULL for focus vessel
- ← *frame* frame of reference flag
- *vel* pointer to variable receiving ground speed vector [m/s in x,y,z]

**Returns:**

Error flag (*false* indicates error)

**Note:**

This method returns the ground speed vector in the requested frame of reference. The ground speed vector is defined as the vessel's velocity vector with respect to a point at the vessel position fixed in the planet's rotating frame of reference.

Valid entries for *frame* are

- FRAME\_GLOBAL: Returns velocity vector in the global frame of reference
- FRAME\_LOCAL: Returns velocity vector in the vessel's local frame of reference
- FRAME\_REFLOCAL: Returns velocity vector in the celestial reference body's local frame of reference
- FRAME\_HORIZON: Returns velocity vector in the local horizon frame (x = longitudinal component, y = vertical component, z = latitudinal component)

**See also:**

[oapiGetGroundspeed](#), [oapiGetAirspeedVector](#), [oapiGetAirspeed](#), [VESSEL::GetGroundspeedVector](#)

**15.45.1.26 OAPIFUNC BOOL oapiGetHeading (OBJHANDLE *hVessel*, double \* *heading*)**

Returns a vessel's heading (against geometric north) calculated for the local horizon plane.

**Parameters:**

- hVessel* vessel handle
- heading* pointer to variable receiving heading value [rad]

**Returns:**

Error flag (*false* on failure)

**Note:**

Unit is radian [rad] 0=north, PI/2=east, etc.

The handle passed to the function must refer to a vessel.

**See also:**

[oapiGetBank](#), [oapiGetPitch](#), [oapiGetAltitude](#)

**15.45.1.27 OAPIFUNC double oapiGetMaxFuelMass (OBJHANDLE *hVessel*)**

Returns maximum fuel capacity of the first propellant resource of a vessel.

**Parameters:**

*hVessel* vessel handle

**Returns:**

Maximum fuel mass [kg]

**Note:**

This function is equivalent to

```
oapiGetPropellantMaxMass (oapiGetPropellantHandle (hVessel, 0))
```

*hVessel* must be a vessel handle. Other object types are invalid.

For multistage configurations, this returns the sum of the max fuel mass of active stages only.

**15.45.1.28 OAPIFUNC BOOL oapiGetPitch (OBJHANDLE *hVessel*, double \**pitch*)**

Returns a vessel's pitch angle w.r.t. the local horizon.

**Parameters:**

*hVessel* vessel handle

*pitch* pointer to variable receiving pitch value

**Returns:**

Error flag (*false* on failure)

**Note:**

Unit is radian [rad]

Returns pitch angle w.r.t. closest planet

The local horizon is the plane whose normal is defined by the distance vector from the planet centre to the vessel.

The handle passed to the function must refer to a vessel.

**See also:**

[oapiGetHeading](#), [oapiGetBank](#), [oapiGetAltitude](#)

**15.45.1.29 OAPIFUNC PROPELLANT\_HANDLE oapiGetPropellantHandle (OBJHANDLE *hVessel*, DWORD *idx*)**

Returns an identifier of a vessel's propellant resource.

**Parameters:**

*hVessel* vessel handle

*idx* propellant resource index (>=0)

**Returns:**

propellant resource id, or NULL if *idx* >= # propellant resources

**15.45.1.30 OAPIFUNC double oapiGetPropellantMass (PROPELLANT\_HANDLE *ph*)**

Returns the current fuel mass [kg] of a propellant resource.

**Parameters:**

*ph* propellant resource identifier

**Returns:**

current fuel mass [kg] of the resource.

**See also:**

[oapiGetPropellantMaxMass](#), [oapiGetPropellantHandle](#)

**15.45.1.31 OAPIFUNC double oapiGetPropellantMaxMass (PROPELLANT\_HANDLE *ph*)**

Returns the maximum capacity [kg] of a propellant resource.

**Parameters:**

*ph* propellant resource identifier

**Returns:**

maximum fuel capacity [kg] of the resource.

**See also:**

[oapiGetPropellantHandle](#), [VESSEL::GetPropellantMaxMass](#)

**15.45.1.32 OAPIFUNC bool oapiSetAttitudeMode (OBJHANDLE *hVessel*, int *mode*)**

Set a vessel's attitude thruster mode.

**Parameters:**

*hVessel* vessel handle

*mode* attitude mode (0=disable, 1=rotational, 2=linear)

**Returns:**

Error flag; *false* indicates failure (requested mode not available)

**Note:**

The handle must refer to a vessel. This function does not support other object types.

**See also:**

[oapiToggleAttitudeMode](#), [oapiGetAttitudeMode](#)

**15.45.1.33 OAPIFUNC void oapiSetEmptyMass (OBJHANDLE *hVessel*, double *mass*)**

Set the empty mass of a vessel (excluding fuel).

**Parameters:**

*hVessel* vessel handle

*mass* empty mass [kg]

**Note:**

Use this function to register structural mass changes, for example as a result of jettisoning a fuel tank, etc.

**15.45.1.34 OAPIFUNC void oapiSetEngineLevel (OBJHANDLE *hVessel*, ENGINETYPE *engine*, double *level*)**

Engage the specified engines.

**Parameters:**

*hVessel* vessel handle

*engine* identifies the engine to be set

*level* engine thrust level [0,1]

**Note:**

Not all vessels support all types of engines.

Setting main thrusters > 0 implies setting retro thrusters to 0 and vice versa.

Setting main thrusters to -level is equivalent to setting retro thrusters to +level and vice versa.

**15.45.1.35 OAPIFUNC bool oapiSetFocusAttitudeMode (int *mode*)**

Set the current focus vessel's attitude thruster mode.

**Parameters:**

*mode* attitude mode (0=disable, 1=rotational, 2=linear)

**Returns:**

Error flag; *false* indicates error (requested mode not available)

**See also:**

[oapiGetFocusAttitudeMode](#), [oapiToggleFocusAttitudeMode](#)

**15.45.1.36 OAPIFUNC int oapiToggleAttitudeMode (OBJHANDLE *hVessel*)**

Flip a vessel's attitude thruster mode between rotational and linear.

**Parameters:**

*hVessel* vessel handle

**Returns:**

The new attitude mode (1=rotational, 2=linear, 0=unchanged disabled)

**Note:**

The handle must refer to a vessel. This function does not support other object types.  
This function flips between linear and rotational, but has no effect if attitude thrusters were disabled.

**See also:**

[oapiSetAttitudeMode](#), [oapiGetAttitudeMode](#)

**15.45.1.37 OAPIFUNC int oapiToggleFocusAttitudeMode ()**

Flip the current focus vessel's attitude thruster mode between rotational and linear.

**Returns:**

The new attitude mode (1=rotational, 2=linear, 0=unchanged disabled)

**Note:**

This function flips between linear and rotational, but has no effect if attitude thrusters were disabled.

**See also:**

[oapiSetFocusAttitudeMode](#), [oapiGetFocusAttitudeMode](#)

**15.46 Coordinate transformations****Functions**

- OAPIFUNC void [oapiGetRotationMatrix](#) (**OBJHANDLE** hObj, **MATRIX3** \*mat)  
*Returns the current rotation matrix of an object.*
- OAPIFUNC void [oapiGlobalToLocal](#) (**OBJHANDLE** hObj, const **VECTOR3** \*glob, **VECTOR3** \*loc)  
*Maps a point from the global frame to a local object frame.*
- OAPIFUNC void [oapiLocalToGlobal](#) (**OBJHANDLE** hObj, const **VECTOR3** \*loc, **VECTOR3** \*glob)  
*Maps a point from a local object frame to the global frame.*
- OAPIFUNC void [oapiEquToLocal](#) (**OBJHANDLE** hObj, double lng, double lat, double rad, **VECTOR3** \*loc)  
*Returns the cartesian position in the local object frame of a point given in equatorial coordinates.*
- OAPIFUNC void [oapiLocalToEqu](#) (**OBJHANDLE** hObj, const **VECTOR3** &loc, double \*lng, double \*lat, double \*rad)  
*Returns the equatorial coordinates of a point given in the local frame of an object.*
- OAPIFUNC void [oapiEquToGlobal](#) (**OBJHANDLE** hObj, double lng, double lat, double rad, **VECTOR3** \*glob)

*Returns the global cartesian position of a point given in equatorial coordinates of an object.*

- OAPIFUNC void [oapiGlobalToEqu](#) (OBJHANDLE *hObj*, const VECTOR3 &*glob*, double \**lng*, double \**lat*, double \**rad*)

*Returns the equatorial coordinates with respect to an object of a point given in the global reference frame.*

- OAPIFUNC double [oapiOrthodome](#) (double *lng1*, double *lat1*, double *lng2*, double *lat2*)

*Returns the angular distance of two points on a sphere.*

### 15.46.1 Function Documentation

#### 15.46.1.1 OAPIFUNC void oapiEquToGlobal (OBJHANDLE *hObj*, double *lng*, double *lat*, double *rad*, VECTOR3 \* *glob*)

Returns the global cartesian position of a point given in equatorial coordinates of an object.

##### Parameters:

- ← *hObj* object handle
- ← *lng* longitude of point [rad]
- ← *lat* latitude of point [rad]
- ← *rad* distance from local object origin [m]
- *glob* point in cartesian coordinates of the global reference frame [**m**]

##### See also:

[oapiGlobalToEqu](#), [oapiEquToLocal](#), [oapiLocalToEqu](#)

#### 15.46.1.2 OAPIFUNC void oapiEquToLocal (OBJHANDLE *hObj*, double *lng*, double *lat*, double *rad*, VECTOR3 \* *loc*)

Returns the cartesian position in the local object frame of a point given in equatorial coordinates.

##### Parameters:

- ← *hObj* object handle
- ← *lng* longitude of point [rad]
- ← *lat* latitude of point [rad]
- ← *rad* distance from local object origin [m]
- *loc* point in cartesian coordinates of the local object frame [**m**]

##### See also:

[oapiLocalToEqu](#), [oapiEquToLocal](#), [oapiGlobalToEqu](#)

**15.46.1.3 OAPIFUNC void oapiGetRotationMatrix (OBJHANDLE *hObj*, MATRIX3 \* *mat*)**

Returns the current rotation matrix of an object.

**Parameters:**

- ← *hObj* object handle
- *mat* rotation matrix

**Note:**

The returned rotation matrix can be used to transform orientations from the local frame of an object to Orbiter's global reference frame (ecliptic and equinox of J2000) and vice versa.

The rotation, defined by matrix R, together with a translation vector t, provides the transformation of a point p between local and global coordinates:

$$\vec{p}_{\text{global}} = \mathbf{R}\vec{p}_{\text{local}} + \vec{t}$$

and

$$\vec{p}_{\text{local}} = \mathbf{R}^T(\vec{p}_{\text{global}} - \vec{t})$$

**See also:**

[VESSEL::GetRotationMatrix](#),  
[mul\(const MATRIX3&,const VECTOR3&\)](#),  
[tmul\(const MATRIX3&,const VECTOR3&\)](#)

**15.46.1.4 OAPIFUNC void oapiGlobalToEqu (OBJHANDLE *hObj*, const VECTOR3 & *glob*, double \* *lng*, double \* *lat*, double \* *rad*)**

Returns the equatorial coordinates with respect to an object of a point given in the global reference frame.

**Parameters:**

- ← *hObj* object handle
- ← *glob* point in global coordinates
- *lng* pointer to variable receiving the longitude value [rad]
- *lat* pointer to variable receiving the latitude value [rad]
- *rad* pointer to variable receiving the radial distance value [m]

**See also:**

[oapiEquToLocal](#), [oapiLocalToEqu](#), [oapiEquToGlobal](#)

**15.46.1.5 OAPIFUNC void oapiGlobalToLocal (OBJHANDLE *hObj*, const VECTOR3 \* *glob*, VECTOR3 \* *loc*)**

Maps a point from the global frame to a local object frame.

**Parameters:**

- ← *hObj* object handle
- ← *glob* point in global coordinates

$\rightarrow \text{loc}$  point mapped into local coordinates

**Note:**

This function maps global point  $\text{glob}$  into the local reference frame of body  $\text{hObj}$ . The transformation is given by

$$\vec{p}_{\text{loc}} = \mathbf{R}_{\text{hObj}}^T(\vec{p}_{\text{glob}} - \vec{p}_{\text{hObj}})$$

where  $\mathbf{R}_{\text{hObj}}$ ,  $\vec{p}_{\text{hObj}}$  are the body's rotation matrix and global position, respectively.

**See also:**

[oapiLocalToGlobal](#), [oapiGetRotationMatrix](#)

#### 15.46.1.6 OAPIFUNC void oapiLocalToEqu (OBJHANDLE $\text{hObj}$ , const VECTOR3 & $\text{loc}$ , double \* $\text{lng}$ , double \* $\text{lat}$ , double \* $\text{rad}$ )

Returns the equatorial coordinates of a point given in the local frame of an object.

**Parameters:**

- $\leftarrow \text{hObj}$  object handle
- $\leftarrow \text{loc}$  point in cartesian coordinates of the local object frame [m]
- $\rightarrow \text{lng}$  pointer to variable receiving the longitude value [rad]
- $\rightarrow \text{lat}$  pointer to variable receiving the latitude value [rad]
- $\rightarrow \text{rad}$  pointer to variable receiving the radial distance value [m]

**See also:**

[oapiEquToLocal](#), [oapiEquToGlobal](#), [oapiGlobalToEqu](#)

#### 15.46.1.7 OAPIFUNC void oapiLocalToGlobal (OBJHANDLE $\text{hObj}$ , const VECTOR3 \* $\text{loc}$ , VECTOR3 \* $\text{glob}$ )

Maps a point from a local object frame to the global frame.

**Parameters:**

- $\leftarrow \text{hObj}$  object handle
- $\leftarrow \text{loc}$  point in local coordinates of the object frame
- $\rightarrow \text{glob}$  point mapped into global coordinates

**Note:**

This function maps point  $\text{loc}$  given in local coordinates of  $\text{hObj}$  into the global reference frame (barycentric ecliptic and equinox of J2000). The transformation is given by

$$\vec{p}_{\text{glob}} = \mathbf{R}_{\text{hObj}}\vec{p}_{\text{loc}} + \vec{p}_{\text{hObj}}$$

where  $\mathbf{R}_{\text{hObj}}$ ,  $\vec{p}_{\text{hObj}}$  are the body's rotation matrix and global position, respectively.

**See also:**

[oapiGlobalToLocal](#), [oapiGetRotationMatrix](#)

#### 15.46.1.8 OAPIFUNC double oapiOrthodome (double *lng1*, double *lat1*, double *lng2*, double *lat2*)

Returns the angular distance of two points on a sphere.

**Parameters:**

- *lng1* longitude value of point 1 [rad]
- *lat1* latitude value of point 1 [rad]
- *lng2* longitude value of point 2 [rad]
- *lat2* latitude value of point 2 [rad]

**Note:**

Given two points on the surface of a sphere, this function returns the orthodome (shortest) angular distance between them.

The shortest surface path between the points is an arc on a great circle containing the two points, and its length is given by  $d = aR$ , where  $a$  is the angular distance returned by oapiOrthodome, and  $R$  is the radius of the sphere.

## 15.47 Camera functions

### Functions

- OAPIFUNC bool [oapiCameraInternal \(\)](#)  
*Returns flag to indicate internal/external camera mode.*
- OAPIFUNC int [oapiCameraMode \(\)](#)  
*Returns the current camera view mode.*
- OAPIFUNC int [oapiCockpitMode \(\)](#)  
*Returns the current cockpit display mode.*
- OAPIFUNC OBJHANDLE [oapiCameraTarget \(\)](#)  
*Returns a handle to the current camera target.*
- OAPIFUNC OBJHANDLE [oapiCameraProxyGbody \(\)](#)  
*Returns celestial body whose surface is closest to the camera.*
- OAPIFUNC void [oapiCameraGlobalPos \(VECTOR3 \\*gpos\)](#)  
*Returns current camera position in global coordinates.*
- OAPIFUNC void [oapiCameraGlobalDir \(VECTOR3 \\*gdir\)](#)  
*Returns current camera direction in global coordinates.*
- OAPIFUNC void [oapiCameraRotationMatrix \(MATRIX3 \\*rmat\)](#)
- OAPIFUNC double [oapiCameraTargetDist \(\)](#)  
*Returns the distance between the camera and its target [m].*
- OAPIFUNC double [oapiCameraAzimuth \(\)](#)  
*Returns the current camera azimuth angle with respect to the target.*

- OAPIFUNC double `oapiCameraPolar ()`  
*Returns the current camera polar angle with respect to the target.*
- OAPIFUNC double `oapiCameraAperture ()`  
*Returns the current camera aperture (the field of view) in rad.*
- OAPIFUNC void `oapiCameraSetAperture (double aperture)`  
*Change the camera aperture (field of view).*
- OAPIFUNC void `oapiCameraScaleDist (double dscale)`  
*Moves the camera closer to the target or further away.*
- OAPIFUNC void `oapiCameraRotAzimuth (double dazimuth)`  
*Rotate the camera around the target (azimuth angle).*
- OAPIFUNC void `oapiCameraRotPolar (double dpolar)`  
*Rotate the camera around the target (polar angle).*
- OAPIFUNC void `oapiCameraSetCockpitDir (double polar, double azimuth, bool transition=false)`  
*Set the camera direction in cockpit mode.*
- OAPIFUNC void `oapiCameraAttach (OBJHANDLE hObj, int mode)`  
*Attach the camera to a new target, or switch between internal and external camera mode.*

### 15.47.1 Function Documentation

#### 15.47.1.1 OAPIFUNC double oapiCameraAperture ()

Returns the current camera aperture (the field of view) in rad.

##### Returns:

`camera aperture [rad]`

##### Note:

Orbiter defines the the aperture as 1/2 of the vertical field of view, between the viewport centre and the top edge of the viewport.

#### 15.47.1.2 OAPIFUNC void oapiCameraAttach (OBJHANDLE *hObj*, int *mode*)

Attach the camera to a new target, or switch between internal and external camera mode.

##### Parameters:

`hObj` handle of the new camera target

`mode` camera mode (0=internal, 1=external, 2=don't change)

**Note:**

If the new target is not a vessel, the camera mode is always set to external, regardless of the value of mode.

**See also:**

[oapiCameraMode](#), [oapiCameraTarget](#)

**15.47.1.3 OAPIFUNC double oapiCameraAzimuth ()**

Returns the current camera azimuth angle with respect to the target.

**Returns:**

Camera azimuth angle [rad]. Value 0 indicates that the camera is behind the target.

**Note:**

This function is useful only in external camera mode. In internal mode, it will always return 0.

**15.47.1.4 OAPIFUNC void oapiCameraGlobalDir (VECTOR3 \* *gdir*)**

Returns current camera direction in global coordinates.

**Parameters:**

*gdir* pointer to vector to receive global camera direction

**See also:**

[oapiCameraGlobalPos](#)

**15.47.1.5 OAPIFUNC void oapiCameraGlobalPos (VECTOR3 \* *gpos*)**

Returns current camera position in global coordinates.

**Parameters:**

*gpos* pointer to vector to receive global camera coordinates

**Note:**

The global coordinate system is the heliocentric ecliptic frame at epoch J2000.0.

**See also:**

[oapiCameraGlobalDir](#)

**15.47.1.6 OAPIFUNC bool oapiCameraInternal ()**

Returns flag to indicate internal/external camera mode.

**Returns:**

*true* indicates an internal camera mode, i.e. the camera is located inside a vessel cockpit. In this case, the camera target is always the current focus object. *false* indicates an external camera mode, i.e. the camera points toward an object from outside. The camera target may be a vessel, planet, spaceport, etc.

**See also:**

[oapiCameraMode](#), [oapiCockpitMode](#)

**15.47.1.7 OAPIFUNC int oapiCameraMode ()**

Returns the current camera view mode.

**Returns:**

Camera mode:

- CAM\_COCKPIT cockpit (internal) mode
- CAM\_TARGETRELATIVE tracking mode (relative direction)
- CAM\_ABSDIRECTION tracking mode (absolute direction)
- CAM\_GLOBALFRAME tracking mode (global frame)
- CAM\_TARGETTOOBJECT tracking mode (target to object)
- CAM\_TARGETFROMOBJECT tracking mode (object to target)
- CAM\_GROUNDOBSERVER ground observer mode

**See also:**

[oapiCameraInternal](#), [VESSEL::GetCameraOffset](#), [VESSEL::GetCameraDefaultDirection](#)

**15.47.1.8 OAPIFUNC double oapiCameraPolar ()**

Returns the current camera polar angle with respect to the target.

**Returns:**

Camera polar angle [rad]. Value 0 indicates that the camera is at the same elevation as the target.

**Note:**

This function is useful only in external camera mode. In internal mode, it will always return 0.

**15.47.1.9 OAPIFUNC void oapiCameraRotAzimuth (double *dazimuth*)**

Rotate the camera around the target (azimuth angle).

**Parameters:**

*dazimuth* change in azimuth angle [rad]

**Note:**

This function is ignored if the camera is in internal mode.

**15.47.1.10 OAPIFUNC void oapiCameraRotPolar (double *dpolar*)**

Rotate the camera around the target (polar angle).

**Parameters:**

*dpolar* change in polar angle [rad]

**Note:**

This function is ignored if the camera is in internal mode.

**15.47.1.11 OAPIFUNC void oapiCameraScaleDist (double *dscale*)**

Moves the camera closer to the target or further away.

**Parameters:**

*dscale* distance scaling factor

**Note:**

Setting *dscale* < 1 will move the camera closer to its target. *dscale* > 1 will move it further away.  
This function is ignored if the camera is in internal mode.

**15.47.1.12 OAPIFUNC void oapiCameraSetAperture (double *aperture*)**

Change the camera aperture (field of view).

**Parameters:**

*aperture* new aperture [rad]

**Note:**

Orbiter restricts the aperture to the range from RAD\*0.1 to RAD\*80 (i. e. field of view between 0.2 and 160 deg. Very wide angles (> 90 deg) and very narrow angles (< 5 deg) should only be used to implement specific optical devices, e.g. telescopes or wide-angle cameras, not for standard observer views.

The Orbiter user interface does not accept apertures > 45 deg or < 5 deg. As soon as the user manipulates the aperture manually, it will be clamped back to the range from 5 to 45 deg.

**15.47.1.13 OAPIFUNC void oapiCameraSetCockpitDir (double *polar*, double *azimuth*, bool *transition* = false)**

Set the camera direction in cockpit mode.

**Parameters:**

*polar* polar angle [rad]

*azimuth* azimuth angle [rad]

*transition* transition flag (see notes)

**Note:**

This function is ignored if the camera is not currently in cockpit mode.

The polar and azimuth angles are relative to the default view direction (see [VES-SEL::SetCameraDefaultDirection\(\)](#))

The requested direction should be within the current rotation ranges (see [VES-SEL::SetCameraRotationRange\(\)](#)), otherwise the result is undefined.

If transition==false, the new direction is set instantaneously; otherwise the camera swings from the current to the new direction (not yet implemented).

**15.47.1.14 OAPIFUNC OBJHANDLE oapiCameraTarget ()**

Returns a handle to the current camera target.

**Returns:**

Handle to the current camera target (i.e. the object the camera is pointing at in external mode, or the handle of the vessel in cockpit mode)

**Note:**

The camera target is not necessarily a vessel, and if it is a vessel, it is not necessarily the focus object (the vessel receiving user input).

**See also:**

[oapiCameraAttach](#)

**15.47.1.15 OAPIFUNC double oapiCameraTargetDist ()**

Returns the distance between the camera and its target [m].

**Returns:**

Distance between camera and camera target [m].

**15.47.1.16 OAPIFUNC int oapiCockpitMode ()**

Returns the current cockpit display mode.

**Returns:**

Cockpit mode:

- COCKPIT\_GENERIC (generic cockpit mode: left+right [MFD](#) and [HUD](#))
- COCKPIT\_PANELS (2D panel mode)
- COCKPIT\_VIRTUAL (virtual cockpit mode)

**Note:**

This function also works if the camera is not currently in cockpit mode.

**See also:**

[oapiCameraInternal](#), [VESSEL::GetCameraOffset](#), [VESSEL::GetCameraDefaultDirection](#)

## 15.48 Functions for planetary bodies

### 15.48.1 Detailed Description

All OBJHANDLE function parameters used in this section must refer to planetary bodies (planets, moons, asteroids, etc.) unless stated otherwise. Invalid handles may lead to crashes.

Currently, the orientation of planetary rotation axes is assumed time-invariant. Precession, nutation and similar effects are not currently simulated.

#### Functions

- OAPIFUNC double `oapiGetPlanetPeriod (OBJHANDLE hPlanet)`  
*Returns the rotation period (the length of a sidereal day) of a planet.*
- OAPIFUNC double `oapiGetPlanetObliquity (OBJHANDLE hPlanet)`  
*Returns the obliquity of the planet's rotation axis (the angle between the rotation axis and the ecliptic zenith).*
- OAPIFUNC double `oapiGetPlanetTheta (OBJHANDLE hPlanet)`  
*Returns the longitude of the ascending node.*
- OAPIFUNC void `oapiGetPlanetObliquityMatrix (OBJHANDLE hPlanet, MATRIX3 *mat)`  
*Returns a rotation matrix which performs the transformation from the planet's tilted coordinates into global coordinates.*
- OAPIFUNC double `oapiGetPlanetCurrentRotation (OBJHANDLE hPlanet)`  
*Returns the current rotation angle of the planet around its axis.*
- OAPIFUNC bool `oapiPlanetHasAtmosphere (OBJHANDLE hPlanet)`  
*Test for existence of planetary atmosphere.*
- OAPIFUNC void `oapiGetPlanetAtmParams (OBJHANDLE hPlanet, double rad, ATMPARAM *prm)`  
*Returns atmospheric parameters as a function of distance from the planet centre.*
- OAPIFUNC void `oapiGetPlanetAtmParams (OBJHANDLE hPlanet, double alt, double lng, double lat, ATMPARAM *prm)`  
*Returns atmospheric parameters of a planet as a function of altitude and geographic position.*
- OAPIFUNC const `ATMCONST * oapiGetPlanetAtmConstants (OBJHANDLE hPlanet)`  
*Returns atmospheric constants for a planet.*
- OAPIFUNC `VECTOR3 oapiGetGroundVector (OBJHANDLE hPlanet, double lng, double lat, int frame=2)`  
*Returns the velocity vector of a surface point.*
- OAPIFUNC `VECTOR3 oapiGetWindVector (OBJHANDLE hPlanet, double lng, double lat, double alt, int frame=0)`  
*Returns the wind velocity at a given position in a planet's atmosphere.*

- OAPIFUNC DWORD [oapiGetPlanetJCoeffCount](#) (**OBJHANDLE hPlanet**)  
*Returns the number of perturbation coefficients defined for a planet.*
- OAPIFUNC double [oapiGetPlanetJCoeff](#) (**OBJHANDLE hPlanet, DWORD n**)  
*Returns a perturbation coefficient for the calculation of a planet's gravitational potential.*

## 15.48.2 Function Documentation

### 15.48.2.1 OAPIFUNC VECTOR3 [oapiGetGroundVector](#) (**OBJHANDLE hPlanet, double lng, double lat, int frame = 2**)

Returns the velocity vector of a surface point.

#### Parameters:

**hPlanet** planet handle  
**lng** longitude [rad]  
**lat** latitude [rad]  
**frame** reference frame flag (see notes)

#### Returns:

surface velocity [**m**]

#### Note:

The *frame* flag can be used to specify the reference frame to which the returned vector refers. The following values are supported:

- 0: surface-relative (relative to local horizon)
- 1: planet-local (relative to local planet frame)
- 2: planet-local non-rotating
- 3: global (maps to global frame and adds planet velocity)

*frame* = 0 and *frame* = 1 are provided for completeness only. They return (0,0,0) by definition.  
*frame* = 2 returns the following vector for a planet with mean radius *R* and rotation period *T*:

$$\vec{v} = \frac{2\pi R}{T} \cos(\text{lat}) \begin{bmatrix} -\sin(\text{lng}) \\ 0 \\ \cos(\text{lng}) \end{bmatrix}$$

*frame* = 3 maps the vector given above into the global frame and adds the planet velocity.

### 15.48.2.2 OAPIFUNC const ATMCONST\* [oapiGetPlanetAtmConstants](#) (**OBJHANDLE hPlanet**)

Returns atmospheric constants for a planet.

#### Parameters:

**hPlanet** planet handle

**Returns:**

pointer to [ATMCONST](#) structure containing atmospheric coefficients for the planet (see notes)

**Note:**

[ATMCONST](#) has the following components:

```
typedef struct {
    double p0;          // pressure at mean radius ('sea level') [Pa]
    double rho0;        // density at mean radius [kg/m3]
    double R;           // specific gas constant [J/(K kg)]
    double gamma;       // ratio of specific heats, c_p/c_v
    double C;           // exponent for pressure equation (temporary)
    double O2pp;        // partial pressure of oxygen
    double altlimit;   // atmosphere altitude limit [m]
    double radlimit;   // radius limit (altlimit + mean radius)
    double horizontalt; // horizon rendering altitude
    VECTOR3 color0;    // sky colour at sea level during daytime
} ATMCONST;
```

If the specified planet does not have an atmosphere, return value is NULL.

**See also:**

[oapiPlanetHasAtmosphere](#), [oapiGetPlanetAtmParams](#)

**15.48.2.3 OAPIFUNC void oapiGetPlanetAtmParams (OBJHANDLE *hPlanet*, double *alt*, double *lng*, double *lat*, ATMPARAM \* *prm*)**

Returns atmospheric parameters of a planet as a function of altitude and geographic position.

**Parameters:**

*hPlanet* planet handle  
*alt* altitude above planet mean radius [m]  
*lng* longitude [rad]  
*lat* latitude [rad]  
*prm* pointer to [ATMPARAM](#) structure receiving parameters

**See also:**

[oapiGetPlanetAtmParams\(OBJHANDLE,double,double,double,ATMPARAM\\*\)](#), [oapiPlanetHasAtmosphere](#), [oapiGetPlanetAtmConstants](#)

**15.48.2.4 OAPIFUNC void oapiGetPlanetAtmParams (OBJHANDLE *hPlanet*, double *rad*, ATMPARAM \* *prm*)**

Returns atmospheric parameters as a function of distance from the planet centre.

**Parameters:**

*hPlanet* planet handle  
*rad* radius from planet centre [m]  
*prm* pointer to [ATMPARAM](#) structure receiving parameters

**Note:**

If the planet has no atmosphere, or if the defined radius is beyond the defined upper atmosphere limit, all parameters are set to 0.

If the atmosphere model is position- as well as altitude-dependent, this function assumes longitude=0 and latitude=0.

[ATMPARAM](#) has the following components:

```
typedef struct {
    double T;           // temperature [K]
    double p;           // pressure [Pa]
    double rho;         // density [kg/m^3]
} ATMPARAM;
```

**See also:**

[oapiGetPlanetAtmParams\(OBJHANDLE,double,double,double,ATMPARAM\\*\)](#), [oapiPlanetHasAtmosphere](#), [oapiGetPlanetAtmConstants](#)

**15.48.2.5 OAPIFUNC double oapiGetPlanetCurrentRotation (OBJHANDLE *hPlanet*)**

Returns the current rotation angle of the planet around its axis.

**Parameters:**

*hPlanet* planet handle

**Returns:**

Rotation angle [rad]

**Note:**

The complete rotation matrix from planet local to global (ecliptic) coordinates is given by

$$R = R_a \begin{bmatrix} \cos \omega & 0 & -\sin \omega \\ 0 & 1 & 0 \\ \sin \omega & 0 & \cos \omega \end{bmatrix}$$

where  $R_a$  is the obliquity matrix as returned by [oapiGetPlanetObliquityMatrix\(\)](#), and  $\omega$  is the rotation angle returned by [oapiGetPlanetCurrentRotation\(\)](#).

**15.48.2.6 OAPIFUNC double oapiGetPlanetJCoeff (OBJHANDLE *hPlanet*, DWORD *n*)**

Returns a perturbation coefficient for the calculation of a planet's gravitational potential.

**Parameters:**

*hPlanet* planet handle

*n* coefficient index

**Returns:**

Perturbation coefficient  $J_{n+2}$

**Note:**

Valid indices  $n$  are 0 to [oapiGetPlanetJCoeffCount\(\)](#)-1

Orbiter calculates the planet's gravitational potential  $U$  for a given distance  $r$  and latitude  $\phi$  by

$$U(r, \phi) = \frac{GM}{r} \left[ 1 - \sum_{n=2}^N J_n \left( \frac{R}{r} \right)^2 P_n(\sin \phi) \right]$$

where  $R$  is the planet's equatorial radius,  $M$  is its mass,  $G$  is the gravitational constant, and  $P_n$  is the Legendre polynomial of order  $n$ .

Orbiter currently considers perturbations to be only a function of latitude (polar), not of longitude.

The first coefficient,  $n = 0$ , returns  $J_2$ , which accounts for the ellipsoid shape of a planet (flattening). Higher perturbation terms are usually small compared to  $J_2$  (and not known for most planets).

**See also:**

[oapiGetPlanetJCoeffCount](#)

**15.48.2.7 OAPIFUNC DWORD oapiGetPlanetJCoeffCount (OBJHANDLE *hPlanet*)**

Returns the number of perturbation coefficients defined for a planet.

Returns the number of perturbation coefficients defined for a planet to describe the latitude-dependent perturbation of its gravitational potential. A return value of 0 indicates that the planet is considered to have a spherically symmetric gravity field.

**Parameters:**

*hPlanet* planet handle

**Returns:**

Number of perturbation coefficients.

**Note:**

Even if a planet defines perturbation coefficients, its gravity perturbation may be ignored, if the user disabled nonspherical gravity sources, or if orbit stabilisation is active at a given time step. Use the [VESSEL::NonsphericalGravityEnabled\(\)](#) function to check if a vessel uses the perturbation terms in the update of its state vectors.

Depending on the distance to the planet, Orbiter may use fewer perturbation terms than defined, if their contribution is negligible:

If  $J_n \left( \frac{R}{r} \right)^n < \epsilon$ ,  $n \geq 2$ , ignore all terms  $\geq n$ ,

where  $R$  is the planet radius,  $r$  is the distance from the planet, and  $J_n$  is the  $n$ - 2nd perturbation term defined for the planet.

Orbiter uses  $\epsilon = 10^{-10}$

**15.48.2.8 OAPIFUNC double oapiGetPlanetObliquity (OBJHANDLE *hPlanet*)**

Returns the obliquity of the planet's rotation axis (the angle between the rotation axis and the ecliptic zenith).

**Parameters:**

*hPlanet* planet handle

**Returns:**

obliquity [rad]

**Note:**

In Orbiter, the ecliptic zenith (at epoch J2000) is the positive y-axis of the global frame of reference.

**See also:**

[oapiGetPlanetPeriod](#), [oapiGetPlanetTheta](#)

**15.48.2.9 OAPIFUNC void oapiGetPlanetObliquityMatrix (OBJHANDLE *hPlanet*, MATRIX3 \* *mat*)**

Returns a rotation matrix which performs the transformation from the planet's tilted coordinates into global coordinates.

**Parameters:**

*hPlanet* planet handle

*mat* pointer to a matrix receiving the rotation data

**Note:**

The returned matrix is given by

$$R_a = \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \varphi & -\sin \varphi \\ 0 & \sin \varphi & \cos \varphi \end{bmatrix}$$

where  $\theta$  is the longitude of the ascending node of the equator, as returned by [oapiGetPlanetTheta\(\)](#), and  $\varphi$  is the obliquity as returned by [oapiGetPlanetObliquity\(\)](#).  $R_a$  does not include the current rotation of the planet around its axis.  $R_a$  is therefore time-independent.

**See also:**

[oapiGetPlanetPeriod](#)

**15.48.2.10 OAPIFUNC double oapiGetPlanetPeriod (OBJHANDLE *hPlanet*)**

Returns the rotation period (the length of a siderial day) of a planet.

**Parameters:**

*hPlanet* planet handle

**Returns:**

planet rotation period [seconds]

**See also:**

[oapiGetPlanetObliquity](#), [oapiGetPlanetTheta](#)

**15.48.2.11 OAPIFUNC double oapiGetPlanetTheta (OBJHANDLE *hPlanet*)**

Returns the longitude of the ascending node.

Returns the longitude of the ascending node of the equatorial plane (denoted by  $q$ ), that is, the angle between the vernal equinox and the ascending node of the equator w.r.t. the ecliptic.

**Parameters:**

*hPlanet* planet handle

**Returns:**

longitude of ascending node of the equator [rad]

**Note:**

For Earth, this function will return 0. (The ascending node of Earth's equatorial plane is the definition of the vernal equinox).

**See also:**

[oapiGetPlanetPeriod](#), [oapiGetPlanetObliquity](#)

**15.48.2.12 OAPIFUNC VECTOR3 oapiGetWindVector (OBJHANDLE *hPlanet*, double *lng*, double *lat*, double *alt*, int *frame* = 0)**

Returns the wind velocity at a given position in a planet's atmosphere.

**Parameters:**

*hPlanet* planet handle

*lng* longitude [rad]

*lat* latitude [rad]

*altitude* above mean planet radius [m]

*frame* reference frame flag (see notes)

**Returns:**

wind velocity vector relative to surface [m]

**Note:**

The *frame* flag can be used to specify the reference frame to which the returned vector refers. The following values are supported:

- 0: surface-relative (relative to local horizon)
- 1: planet-local (relative to local planet frame)
- 2: planet-local non-rotating (as 1, but adds the surface velocity, see [oapiGetGroundVector](#))
- 3: global (maps to global frame and adds planet velocity)

**Warning:**

Local wind velocities are not currently implemented. The surface-relative wind velocity is always (0,0,0). To ensure forward compatibility, plugins should not rely on this limitation, but use this function instead.

**15.48.2.13 OAPIFUNC bool oapiPlanetHasAtmosphere (OBJHANDLE *hPlanet*)**

Test for existence of planetary atmosphere.

**Parameters:**

*hPlanet* planet handle

**Returns:**

*true* if an atmosphere has been defined for the planet, *false* otherwise.

**See also:**

[oapiGetPlanetAtmParams](#)

**15.49 Surface base interface****Functions**

- OAPIFUNC OBJHANDLE [oapiGetBasePlanet \(OBJHANDLE hBase\)](#)  
*Returns a handle for the planet/moon the given base is located on.*
- OAPIFUNC void [oapiGetBaseEquPos \(OBJHANDLE hBase, double \\*lng, double \\*lat, double \\*rad=0\)](#)  
*Returns the equatorial coordinates (longitude, latitude and radius) of the location of a surface base.*
- OAPIFUNC DWORD [oapiGetBasePadCount \(OBJHANDLE hBase\)](#)  
*Returns the number of VTOL landing pads owned by the base.*
- OAPIFUNC bool [oapiGetBasePadEquPos \(OBJHANDLE hBase, DWORD pad, double \\*lng, double \\*lat, double \\*rad=0\)](#)  
*Returns the equatorial coordinates (longitude, latitude and radius) of the location of a VTOL landing pad.*
- OAPIFUNC bool [oapiGetBasePadStatus \(OBJHANDLE hBase, DWORD pad, int \\*status\)](#)  
*Returns the status of a VTOL landing pad (free, occupied or cleared).*
- OAPIFUNC NAVHANDLE [oapiGetBasePadNav \(OBJHANDLE hBase, DWORD pad\)](#)  
*Returns a handle to the ILS transmitter of a VTOL landing pad, if available.*

**15.49.1 Function Documentation****15.49.1.1 OAPIFUNC void oapiGetBaseEquPos (OBJHANDLE *hBase*, double \* *lng*, double \* *lat*, double \* *rad* = 0)**

Returns the equatorial coordinates (longitude, latitude and radius) of the location of a surface base.

**Parameters:**

*hBase* surface base handle

*lng* pointer to variable to receive longitude value [rad]

*lat* pointer to variable to receive latitude value [rad]

*rad* pointer to variable to receive radius value [m]

**Note:**

hBase must be a valid base handle (e.g. from [oapiGetBaseByName\(\)](#))

The radius pointer can be omitted if not required.

Currently, rad will always return the planet mean radius.

#### 15.49.1.2 OAPIFUNC DWORD oapiGetBasePadCount (OBJHANDLE *hBase*)

Returns the number of VTOL landing pads owned by the base.

**Parameters:**

*hBase* surface base handle

**Returns:**

Number of landing pads

**Note:**

hBase must be a valid base handle (e.g. from [oapiGetBaseByName\(\)](#))

This function only counts VTOL pads, not runways.

#### 15.49.1.3 OAPIFUNC bool oapiGetBasePadEquPos (OBJHANDLE *hBase*, DWORD *pad*, double \* *lng*, double \* *lat*, double \* *rad* = 0)

Returns the equatorial coordinates (longitude, latitude and radius) of the location of a VTOL landing pad.

**Parameters:**

*hBase* surface base handle

*pad* pad index

*lng* pointer to variable to receive longitude value [rad]

*lat* pointer to variable to receive latitude value [rad]

*rad* pointer to variable to receive radius value [m]

**Returns:**

*false* indicates failure (pad index out of range). In that case, the return values are undefined.

**Note:**

hBase must be a valid base handle (e.g. from [oapiGetBaseByName\(\)](#))

0 <= pad < [oapiGetBasePadCount\(\)](#) is required.

The radius pointer can be omitted if not required.

**15.49.1.4 OAPIFUNC NAVHANDLE oapiGetBasePadNav (OBJHANDLE *hBase*, DWORD *pad*)**

Returns a handle to the ILS transmitter of a VTOL landing pad, if available.

**Parameters:**

*hBase* surface base handle  
*pad* pad index

**Returns:**

Handle of a ILS transmitter, or NULL if the pad index is out of range or the pad has no ILS.

**Note:**

*hBase* must be a valid base handle (e.g. from [oapiGetBaseByName\(\)](#))  
0 <= *pad* < [oapiGetBasePadCount\(\)](#) is required.

**15.49.1.5 OAPIFUNC bool oapiGetBasePadStatus (OBJHANDLE *hBase*, DWORD *pad*, int \* *status*)**

Returns the status of a VTOL landing pad (free, occupied or cleared).

**Parameters:**

*hBase* surface base handle  
*pad* pad index  
*status* pointer to variable to receive pad status

**Returns:**

*false* indicates failure (pad index out of range)

**Note:**

*hBase* must be a valid base handle (e.g. from [oapiGetBaseByName\(\)](#))  
0 <= *pad* < [oapiGetBasePadCount\(\)](#) is required.  
*status* can be one of the following:  
0 = pad is free  
1 = pad is occupied  
2 = pad is cleared for an incoming vessel

**15.49.1.6 OAPIFUNC OBJHANDLE oapiGetBasePlanet (OBJHANDLE *hBase*)**

Returns a handle for the planet/moon the given base is located on.

**Parameters:**

*hBase* base handle

**Returns:**

Planet handle, or NULL if the base was not recognised.

**See also:**

[oapiGetBaseByIndex](#), [oapiGetBaseByName](#)

## 15.50 Time functions

### Functions

- OAPIFUNC double [oapiGetSimTime \(\)](#)  
*Retrieve simulation time (in seconds) since simulation start.*
- OAPIFUNC double [oapiGetSimStep \(\)](#)  
*Retrieve length of last simulation time step (from previous to current frame) in seconds.*
- OAPIFUNC double [oapiGetSysTime \(\)](#)  
*Retrieve system (real) time since simulation start.*
- OAPIFUNC double [oapiGetSysStep \(\)](#)  
*Retrieve length of last system time step in seconds.*
- OAPIFUNC double [oapiGetSimMJD \(\)](#)  
*Retrieve absolute time measure (Modified Julian Date) for current simulation state.*
- OAPIFUNC double [oapiGetSysMJD \(\)](#)  
*Retrieve the current computer system time in Modified Julian Date (MJD) format.*
- OAPIFUNC bool [oapiSetSimMJD \(double mjd, int pmode=0\)](#)  
*Set the current simulation time. The simulation session performs a jump to the new time.*
- OAPIFUNC double [oapiTime2MJD \(double simt\)](#)  
*Convert a simulation up time value into a Modified Julian Date.*
- OAPIFUNC double [oapigetTimeAcceleration \(\)](#)  
*Returns simulation time acceleration factor.*
- OAPIFUNC void [oapiSetTimeAcceleration \(double warp\)](#)  
*Set the simulation time acceleration factor.*
- OAPIFUNC double [oapiGetFrameRate \(\)](#)  
*Returns current simulation frame rate (frames/sec).*
- OAPIFUNC bool [oapiGetPause \(\)](#)  
*Returns the current simulation pause state.*
- OAPIFUNC void [oapiSetPause \(bool pause\)](#)  
*Sets the simulation pause state.*

### 15.50.1 Function Documentation

#### 15.50.1.1 OAPIFUNC double oapiGetFrameRate ()

Returns current simulation frame rate (frames/sec).

##### Returns:

Current frame rate (fps)

**15.50.1.2 OAPIFUNC bool oapiGetPause ()**

Returns the current simulation pause state.

**Returns:**

*true* if simulation is currently paused, *false* if it is running.

**See also:**

[oapiSetPause](#)

**15.50.1.3 OAPIFUNC double oapiGetSimMJD ()**

Retrieve absolute time measure (Modified Julian Date) for current simulation state.

**Returns:**

Current Modified Julian Date (days)

**Note:**

Orbiter defines the Modified Julian Date (MJD) as JD - 240 0000.5, where JD is the Julian Date. JD is the interval of time in mean solar days elapsed since 4713 BC January 1 at Greenwich mean noon.

**See also:**

[oapiSetSimMJD](#), [oapiGetSimTime](#)

**15.50.1.4 OAPIFUNC double oapiGetSimStep ()**

Retrieve length of last simulation time step (from previous to current frame) in seconds.

**Returns:**

Simulation time step (seconds)

**Note:**

This parameter is useful for numerical (finite difference) calculation of time derivatives.

**15.50.1.5 OAPIFUNC double oapiGetSimTime ()**

Retrieve simulation time (in seconds) since simulation start.

**Returns:**

Simulation up time (seconds)

**Note:**

Since the simulation up time depends on the simulation start time, this parameter is useful mainly for time differences. To get an absolute time parameter, use [oapiGetSimMJD\(\)](#).

**15.50.1.6 OAPIFUNC double oapiGetSysMJD ()**

Retrieve the current computer system time in Modified Julian Date (MJD) format.

**Returns:**

Computer system time in MJD format

**Note:**

The returned value is the UTC time obtained from the computer system clock, plus dt=66.184 seconds to map from UTC to TDB (Barycentric Dynamical Time) used internally by Orbiter. The dt offset was not added in previous Orbiter releases.

**See also:**

[oapiGetSysTime](#)

**15.50.1.7 OAPIFUNC double oapiGetSysStep ()**

Retrieve length of last system time step in seconds.

**Returns:**

System time step (seconds)

**Note:**

Unlike [oapiGetSimStep\(\)](#), this function does not include the time compression factor. It is useful to control actions which do not depend on the simulation time acceleration.

**15.50.1.8 OAPIFUNC double oapiGetSysTime ()**

Retrieve system (real) time since simulation start.

**Returns:**

Real-time simulation up time (seconds)

**Note:**

This function measures the real time elapsed since the simulation was started. Unlike [oapiGetSimTime\(\)](#), it doesn't take into account time acceleration.

**See also:**

[oapiGetSysMJD](#)

**15.50.1.9 OAPIFUNC double oapiGetTimeAcceleration ()**

Returns simulation time acceleration factor.

**Returns:**

time acceleration factor

**Note:**

This function will not return 0 when the simulation is paused. Instead it will return the acceleration factor at which the simulation will resume when unpause. Use oapiGetPause to obtain the pause/resume state.

**See also:**

[oapiSetTimeAcceleration](#)

**15.50.1.10 OAPIFUNC void oapiSetPause (bool *pause*)**

Sets the simulation pause state.

**Parameters:**

*pause* *true* to pause the simulation, *false* to resume.

**See also:**

[oapiGetPause](#)

**15.50.1.11 OAPIFUNC bool oapiSetSimMJD (double *mjd*, int *pmode* = 0)**

Set the current simulation time. The simulation session performs a jump to the new time.

**Parameters:**

*mjd* new simulation time

*pmode* vessel propagation modes (see notes)

**Returns:**

Currently this function always returns *true*.

**Note:**

The new time can be set before or after the current simulation time.

Deterministic objects (planets controlled by Keplerian elements or perturbation code) are propagated directly. Vessels are propagated according to pmode, which can be a combination of

Orbital vessels	.
PROP_ORBITAL_ELEMENTS	Move the vessel along its current orbital trajectory, assuming that no forces other than the central body's gravitational force are acting on the vessel.
PROP_ORBITAL_FIXEDSTATE	Keep the vessel's relative position and velocity with respect to the central body fixed in a non-rotating frame.
PROP_ORBITAL_FIXEDSURF	Keep the vessel's position velocity and attitude fixed relative to the planet surface.
Suborbital vessels	.
PROP_SORBITAL_ELEMENTS	PROP_ORBITAL_ELEMENTS
PROP_SORBITAL_FIXEDSTATE	PROP_ORBITAL_FIXEDSTATE
PROP_SORBITAL_FIXEDSURF	PROP_ORBITAL_FIXEDSURF
PROP_SORBITAL_DESTROY	Destroy any suborbital vessels (i.e. assume that the vessels impacted on the ground during time propagation).

*pmode* can be a bitwise combination of one of the orbital and one of the suborbital modes. Default is propagation along osculating elements for both.

See also:

[oapiGetSimMJD](#)

#### 15.50.1.12 OAPIFUNC void oapiSetTimeAcceleration (double *warp*)

Set the simulation time acceleration factor.

Parameters:

*warp* new time acceleration factor

Note:

Warp factors will be clamped to the valid range [1,1000000]. If the new warp factor is different from the previous one, all DLLs (including the one that called [oapiSetTimeAcceleration\(\)](#)) will be sent a [opcTimeAccChanged\(\)](#) message.

See also:

[oapiGetTimeAcceleration](#)

#### 15.50.1.13 OAPIFUNC double oapiTime2MJD (double *simt*)

Convert a simulation up time value into a Modified Julian Date.

Parameters:

*simt* simulation time (seconds)

Returns:

Modified Julian Date (MJD) corresponding to simt.

## 15.51 Navigation radio transmitter functions

Functions

- OAPIFUNC void [oapiGetNavPos](#) (NAVHANDLE hNav, VECTOR3 \*gpos)  
*Returns the current position of a NAV transmitter (in global coordinates, i.e. heliocentric ecliptic).*
- OAPIFUNC DWORD [oapiGetNavChannel](#) (NAVHANDLE hNav)  
*Returns the channel number of a NAV transmitter.*
- OAPIFUNC float [oapiGetNavFreq](#) (NAVHANDLE hNav)  
*Returns the frequency of a NAV transmitter.*
- OAPIFUNC double [oapiGetNavSignal](#) (NAVHANDLE hNav, const VECTOR3 &gpos)  
*Returns the signal strength of a transmitter at a given position.*

- OAPIFUNC float [oapiGetNavRange \(NAVHANDLE hNav\)](#)  
*Returns the range of a NAV transmitter.*
- OAPIFUNC DWORD [oapiGetNavType \(NAVHANDLE hNav\)](#)  
*Returns the type id of a NAV transmitter.*
- OAPIFUNC int [oapiGetNavData \(NAVHANDLE hNav, NAVDATA \\*data\)](#)  
*Returns information about a NAV transmitter.*
- OAPIFUNC int [oapiGetNavDescr \(NAVHANDLE hNav, char \\*descr, int maxlen\)](#)  
*Returns a descriptive string for a NAV transmitter.*
- OAPIFUNC bool [oapiNavInRange \(NAVHANDLE hNav, const VECTOR3 &gpos\)](#)  
*Determines whether a given global coordinate is within the range of a NAV transmitter.*

### 15.51.1 Function Documentation

#### 15.51.1.1 OAPIFUNC DWORD oapiGetNavChannel (NAVHANDLE *hNav*)

Returns the channel number of a NAV transmitter.

##### Parameters:

*hNav* NAV transmitter handle

##### Returns:

channel number

##### Note:

Channel numbers range from 0 to 639.

To convert a channel number ch into a frequency, use  $f = (108.0 + 0.05 \text{ ch}) \text{ MHz}$

##### See also:

[oapiGetNavData](#), [oapiGetNavFreq](#), [oapiGetNavRange](#), [oapiGetNavPos](#), [oapiGetNavType](#)

#### 15.51.1.2 OAPIFUNC int oapiGetNavData (NAVHANDLE *hNav*, NAVDATA \* *data*)

Returns information about a NAV transmitter.

##### Parameters:

$\leftarrow$  *hNav* NAV transmitter handle

$\rightarrow$  *data* pointer to [NAVDATA](#) structure receiving transmitter data

##### Returns:

Error flag. Currently always returns 0.

##### Note:

On call, *data* must point to a [NAVDATA](#) variable.

**See also:**

[NAVDATA](#), [oapiGetNavType](#)

**15.51.1.3 OAPIFUNC int oapiGetNavDescr (NAVHANDLE *hNav*, char \* *descr*, int *maxlen*)**

Returns a descriptive string for a NAV transmitter.

**Parameters:**

*hNav* NAV transmitter handle  
*descr* pointer to string receiving description  
*maxlen* string buffer length

**Returns:**

Number of characters returned (excluding terminating NULL character). If maxlen was not sufficient to store the complete description, the return value is negative.

**Note:**

This function fills string *descr* with a description of the NAV radio transmitter of lenght  $\leq maxlen$ . If the buffer length is greater than required for the description, a NULL character is appended. The description format for the different transmitter types is as follows:

VOR	"VOR <id>"	where <id> is a 3-4 letter sequence
VTOL	"VTOL Pad-<#> <base>"	where <#> is the pad number, and <base> is the base name
ILS	"ILS Rwy <#> <base>"	where <#> is the runway id, and <base> is the base name
IDS	"IDS D-<#> <vessel>"	where <#> is the dock number, and <vessel> is the vessel name
XPDR	"XPDR <vessel>"	where <vessel> is the vessel name

**15.51.1.4 OAPIFUNC float oapiGetNavFreq (NAVHANDLE *hNav*)**

Returns the frequency of a NAV transmitter.

**Parameters:**

*hNav* NAV transmitter handle

**Returns:**

Transmitter frequency [MHz]

**Note:**

In Orbiter, NAV transmitter frequencies range from 108.0 to 139.95 MHz and are incremented in 0.05 MHz steps.

**See also:**

[oapiGetNavData](#), [oapiGetNavChannel](#), [oapiGetNavRange](#), [oapiGetNavPos](#), [oapiGetNavType](#)

**15.51.1.5 OAPIFUNC void oapiGetNavPos (NAVHANDLE *hNav*, VECTOR3 \* *gpos*)**

Returns the current position of a NAV transmitter (in global coordinates, i.e. heliocentric ecliptic).

**Parameters:**

*hNav* NAV transmitter handle  
*gpos* pointer to variable to receive global position

**See also:**

[oapiGetNavRange](#), [oapiGetNavType](#), [oapiNavInRange](#)

**15.51.1.6 OAPIFUNC float oapiGetNavRange (NAVHANDLE *hNav*)**

Returns the range of a NAV transmitter.

**Parameters:**

*hNav* NAV transmitter handle

**Returns:**

Transmitter range [m]

**Note:**

A NAV receiver will only receive a signal when within the range of a transmitter.  
Variable receiver sensitivity is not currently implemented.  
Shadowing of a transmitter by obstacles between transmitter and receiver is not currently implemented.  
Because the range of the transmitter depends on receiver gain as well as transmitter power, the range is not strictly a property of the transmitter. It is preferred to calculate the range for a given receiver gain by using the [oapiGetNavData](#) or [oapiGetNavSignal](#) functions.

**See also:**

[oapiGetNavData](#), [oapiGetNavSignal](#), [oapiGetNavPos](#), [oapiGetNavType](#), [oapiNavInRange](#)

**15.51.1.7 OAPIFUNC double oapiGetNavSignal (NAVHANDLE *hNav*, const VECTOR3 & *gpos*)**

Returns the signal strength of a transmitter at a given position.

**Parameters:**

*hNav* transmitter handle  
*gpos* global position

**Returns:**

Signal strength in arbitrary units

**Note:**

The transmitter signal strength drops off with the square of distance to the transmitter. The units are chosen so that a 'default' receiver will be able to detect signals above a strength of 1.

**See also:**

[oapiGetNavData](#), [oapiGetNavRange](#)

**15.51.1.8 OAPIFUNC DWORD oapiGetNavType (NAVHANDLE *hNav*)**

Returns the type id of a NAV transmitter.

**Parameters:**

*hNav* NAV transmitter handle

**Returns:**

transmitter type identifier

**Note:**

The following transmitter types are currently supported:

- TRANSMITTER\_VOR (omnidirectional beacon)
- TRANSMITTER\_VTOL (launchpad homing beacon)
- TRANSMITTER\_ILS (instrument landing system)
- TRANSMITTER\_IDS (instrument docking system)
- TRANSMITTER\_XPDR (transponder)

**See also:**

[oapiGetNavData](#), [oapiGetNavDescr](#)

**15.51.1.9 OAPIFUNC bool oapiNavInRange (NAVHANDLE *hNav*, const VECTOR3 & *gpos*)**

Determines whether a given global coordinate is within the range of a NAV transmitter.

**Parameters:**

*hNav* NAV transmitter handle

*gpos* Global coordinates [m,m,m] of a point (cartesian heliocentric ecliptic)

**Returns:**

*true* if the point is within range of the transmitter.

## 15.52 Script interpreter functions

### Functions

- OAPIFUNC INTERPRETERHANDLE [oapiCreateInterpreter](#) ()  
*Returns a handle to a new interpreter instance.*
- OAPIFUNC int [oapiDellInterpreter](#) (INTERPRETERHANDLE *hInterp*)  
*Delete an interpreter instance.*
- OAPIFUNC bool [oapiExecScriptCmd](#) (INTERPRETERHANDLE *hInterp*, const char \**cmd*)  
*Executes a script command in an interpreter instance.*
- OAPIFUNC bool [oapiAsyncScriptCmd](#) (INTERPRETERHANDLE *hInterp*, const char \**cmd*)  
*Passes a command to an interpreter instance for execution.*
- OAPIFUNC lua\_State \* [oapiGetLua](#) (INTERPRETERHANDLE *hInterp*)

### 15.52.1 Function Documentation

#### 15.52.1.1 OAPIFUNC bool oapiAsyncScriptCmd (INTERPRETERHANDLE *hInterp*, const char \* *cmd*)

Passes a command to an interpreter instance for execution.

##### Parameters:

*hInterp* interpreter handle  
*cmd* Lua command to be executed

##### Returns:

*false* on error (interpreter library not found, or command error)

##### Note:

This function returns immediately. The command is executed during the next postStep cycle. If more asynchronous commands are issued before execution starts, they are appended to the execution list. If the interpreter receives a synchronous request (oapiExecScriptCmd) before the asynchronous commands are executed, the synchronous command is executed immediately, while the asynchronous requests continue waiting.

##### See also:

[oapiExecScriptCmd](#), [oapiCreateInterpreter](#), [oapiDelInterpreter](#)

#### 15.52.1.2 OAPIFUNC INTERPRETERHANDLE oapiCreateInterpreter ()

Returns a handle to a new interpreter instance.

##### Note:

The interpreter can subsequently be used to execute commands and scripts.

##### See also:

[oapiDelInterpreter](#), [oapiExecScriptCmd](#)

#### 15.52.1.3 OAPIFUNC int oapiDelInterpreter (INTERPRETERHANDLE *hInterp*)

Delete an interpreter instance.

##### Parameters:

*hInterp* interpreter handle

##### Note:

After the interpreter instance has been deleted, the handle becomes invalid and must not be used any more.

If the interpreter was executing a background script, the execution is terminated when the interpreter is deleted.

##### See also:

[oapiCreateInterpreter](#), [oapiExecScriptCmd](#)

**15.52.1.4 OAPIFUNC bool oapiExecScriptCmd (INTERPRETERHANDLE *hInterp*, const char \* *cmd*)**

Executes a script command in an interpreter instance.

**Parameters:**

- hInterp* interpreter handle
- cmd* Lua command to be executed

**Returns:**

*false* on error (interpreter library not found, or command error)

**Note:**

This function returns as soon as the command has been executed.

**See also:**

[oapiAsyncScriptCmd](#), [oapiCreateInterpreter](#), [oapiDelInterpreter](#)

## 15.53 Visual and mesh functions

**Typedefs**

- **typedef void(\* LoadMeshClbkFunc )(MESHHANDLE hMesh, bool firstload)**  
*Callback function used by [oapiLoadMeshGlobal\(const char\\*,LoadMeshClbkFunc\)](#).*

**Functions**

- **OAPIFUNC VISHANDLE \* oapiObjectVisualPtr (OBJHANDLE hObject)**  
*Returns a pointer storing the objects visual handle.*
- **OAPIFUNC MESHHANDLE oapiLoadMesh (const char \*fname)**  
*Loads a mesh from file and returns a handle to it.*
- **OAPIFUNC const MESHHANDLE oapiLoadMeshGlobal (const char \*fname)**  
*Retrieves a mesh handle from the global mesh manager.*
- **OAPIFUNC const MESHHANDLE oapiLoadMeshGlobal (const char \*fname, LoadMeshClbkFunc fClbk)**  
*Retrieves a mesh handle from the global mesh manager.*
- **OAPIFUNC MESHHANDLE oapiCreateMesh (DWORD ngrp, MESHGROUP \*grp)**  
*Creates a new mesh from a list of mesh group definitions.*
- **OAPIFUNC void oapiDeleteMesh (MESHHANDLE hMesh)**  
*Removes a mesh from memory.*
- **OAPIFUNC DWORD oapiMeshGroupCount (MESHHANDLE hMesh)**  
*Returns the number of mesh groups defined in a mesh.*

- OAPIFUNC MESHGROUP \* oapiMeshGroup (**MESHHANDLE** hMesh, **DWORD** idx)  
*Returns a pointer to the group specification of a mesh group.*
- OAPIFUNC MESHGROUP \* oapiMeshGroup (**DEVMESHHANDLE** hMesh, **DWORD** idx)
- OAPIFUNC MESHGROUPEX \* oapiMeshGroupEx (**MESHHANDLE** hMesh, **DWORD** idx)
- OAPIFUNC **DWORD** oapiAddMeshGroup (**MESHHANDLE** hMesh, **MESHGROUP** \*grp)
- OAPIFUNC **bool** oapiAddMeshGroupBlock (**MESHHANDLE** hMesh, **DWORD** grpidx, const **NTVERTEX** \*vtx, **DWORD** nvtx, const **WORD** \*idx, **DWORD** nidx)
- OAPIFUNC **int** oapiEditMeshGroup (**MESHHANDLE** hMesh, **DWORD** grpidx, **GROUPEDITSPEC** \*ges)  
*Modify mesh group data.*
- OAPIFUNC **int** oapiEditMeshGroup (**DEVMESHHANDLE** hMesh, **DWORD** grpidx, **GROUPEDITSPEC** \*ges)
- OAPIFUNC **DWORD** oapiMeshTextureCount (**MESHHANDLE** hMesh)  
*Returns the number of textures associated with a mesh.*
- OAPIFUNC **SURFHANDLE** oapiGetTextureHandle (**MESHHANDLE** hMesh, **DWORD** texidx)  
*Retrieve a surface handle for a mesh texture.*
- OAPIFUNC **SURFHANDLE** oapiLoadTexture (const **char** \*fname, **bool** dynamic=false)  
*Load a texture from a file.*
- OAPIFUNC **void** oapiReleaseTexture (**SURFHANDLE** hTex)  
*Release a texture.*
- OAPIFUNC **bool** oapiSetTexture (**MESHHANDLE** hMesh, **DWORD** texidx, **SURFHANDLE** tex)  
*Replace a mesh texture.*
- OAPIFUNC **bool** oapiSetTexture (**DEVMESHHANDLE** hMesh, **DWORD** texidx, **SURFHANDLE** tex)
- OAPIFUNC **DWORD** oapiMeshMaterialCount (**MESHHANDLE** hMesh)  
*Returns the number of materials defined in the mesh.*
- OAPIFUNC **MATERIAL** \* oapiMeshMaterial (**MESHHANDLE** hMesh, **DWORD** idx)  
*Returns a pointer to a material specification in the material list of the mesh.*
- OAPIFUNC **DWORD** oapiAddMaterial (**MESHHANDLE** hMesh, **MATERIAL** \*mat)  
*Add a material definition to a mesh.*
- OAPIFUNC **bool** oapiDeleteMaterial (**MESHHANDLE** hMesh, **DWORD** idx)  
*Delete a material definition from the mesh.*
- OAPIFUNC **int** oapiSetMaterial (**DEVMESHHANDLE** hMesh, **DWORD** matidx, const **MATERIAL** \*mat)  
*Reset the properties of a mesh material.*
- OAPIFUNC **bool** oapiSetMeshProperty (**MESHHANDLE** hMesh, **DWORD** property, **DWORD** value)

*Set custom properties for a mesh.*

- OAPIFUNC bool [oapiSetMeshProperty](#) (DEVMESHHANDLE hMesh, DWORD property, DWORD value)

*Set custom properties for a device-specific mesh.*

- OAPIFUNC void [oapiParticleSetLevelRef](#) (PSTREAM\_HANDLE ph, double \*lvl)

*Reset the reference pointer used by the particle stream to calculate the intensity (opacity) of the generated particles.*

### 15.53.1 Typedef Documentation

#### 15.53.1.1 **typedef void(\* LoadMeshClbkFunc)(MESHHANDLE hMesh, bool firstload)**

Callback function used by [oapiLoadMeshGlobal\(const char\\*,LoadMeshClbkFunc\)](#).

##### Parameters:

**hMesh** mesh handle

**firstload** flag indicating if the mesh has been loaded for the first time

##### Note:

If firstload==false, the mesh had already been loaded previously. In this case, the mesh is not re-loaded, and the returned handle points to the previously loaded mesh.

### 15.53.2 Function Documentation

#### 15.53.2.1 **OAPIFUNC DWORD oapiAddMaterial (MESHHANDLE hMesh, MATERIAL \* mat)**

Add a material definition to a mesh.

##### Parameters:

**hMesh** mesh handle

**mat** pointer to material definition

##### Returns:

Material index in the mesh.

##### Note:

The material is appended to the mesh material list.

##### See also:

[oapiMeshMaterial](#), [oapiDeleteMaterial](#), [oapiMeshMaterialCount](#)

**15.53.2.2 OAPIFUNC MESHHANDLE oapiCreateMesh (DWORD *ngrp*, MESHGROUP \* *grp*)**

Creates a new mesh from a list of mesh group definitions.

**Parameters:**

*ngrp* number of groups in the list  
*grp* list of mesh groups

**Returns:**

Handle for the newly created mesh.

**Note:**

Orbiter performs a deep copy of the group definitions passed to the functions. Therefore it is admissible to pass the groups as variables with local scope. If the mesh groups were dynamically allocated, they should be deallocated by the caller after use.

**15.53.2.3 OAPIFUNC bool oapiDeleteMaterial (MESHHANDLE *hMesh*, DWORD *idx*)**

Delete a material definition from the mesh.

**Parameters:**

*hMesh* mesh handle  
*idx* material index ( $\geq 0$ )

**Returns:**

*false* indicates failure (index out of range)

**Note:**

This function adjusts all mesh group material indices to account for the modified material table. Any groups that referenced the deleted material are reset to material 0 (default material).

**See also:**

[oapiMeshMaterial](#), [oapiAddMaterial](#), [oapiMeshMaterialCount](#)

**15.53.2.4 OAPIFUNC void oapiDeleteMesh (MESHHANDLE *hMesh*)**

Removes a mesh from memory.

**Parameters:**

*hMesh* mesh handle

**15.53.2.5 OAPIFUNC int oapiEditMeshGroup (MESHHANDLE *hMesh*, DWORD *grpidx*, GROUPEDITSPEC \* *ges*)**

Modify mesh group data.

**Parameters:**

*hMesh* mesh handle  
*grpidx* mesh group index  
*ges* replacement/modification data for the group

**Returns:**

0 on success, or error code

**Note:**

This function allows to modify a mesh group, by replacing vertex data, or group flags. It should not be used to apply a linear transformation to the entire group (use [VES-SEL::MeshgroupTransform](#) instead), because such transformations are usually implemented by defining a transformation matrix instead of editing the vertex positions directly. This version operates on device-independent meshes, e.g. mesh templates. `oapiEditMeshGroup` should be used in preference to [oapiMeshGroup](#), because it is more likely to be supported by external graphics engines.

**See also:**

[oapiEditMeshGroup\(DEVMESSHANDLE,DWORD,GROUPEDITSPEC\\*\)](#)

**15.53.2.6 OAPIFUNC SURFHANDLE oapiGetTextureHandle (MESHHANDLE *hMesh*, DWORD *texidx*)**

Retrieve a surface handle for a mesh texture.

**Parameters:**

*hMesh* mesh handle  
*texidx* texture index ( $>=1$ )

**Returns:**

surface handle

**Note:**

This function can be used for dynamically updating textures during the simulation. the texture index is given by the order in which the textures appear in the texture list at the end of the mesh file. Important: Any textures which are to be dynamically modified should be listed with the "D" flag ("dynamic") in the mesh file. This causes Orbiter to decompress the texture when it is loaded. Blitting operations to compressed surfaces is very inefficient on most graphics hardware.

**15.53.2.7 OAPIFUNC MESHHANDLE oapiLoadMesh (const char \**fname*)**

Loads a mesh from file and returns a handle to it.

**Parameters:**

*fname* mesh file name

**Returns:**

Handle to the loaded mesh. (NULL indicates load error)

**Note:**

The file name should not contain a path or file extension. Orbiter appends extension .msh and searches in the default mesh directory.

Meshes should be deallocated with oapiDeleteMesh when no longer needed.

**See also:**

[oapiDeleteMesh](#), [VESSEL::AddMesh](#)

**15.53.2.8 OAPIFUNC const MESHHANDLE oapiLoadMeshGlobal (const char \**fname*, Load-MeshClbkFunc *fClbk*)**

Retrieves a mesh handle from the global mesh manager.

**Parameters:**

*fname* mesh file name

*fClbk* Callback function for mesh modification

**Returns:**

mesh handle

**Note:**

This function is identical to [oapiLoadMeshGlobal\(const char\\*\)](#), except that it invokes the callback function immediately after loading the mesh. This is important in combination with external graphics clients, because Orbiter hands the loaded mesh on to the client for conversion to a device-specific format. The callback function is invoked before the mesh is passed to the graphics client. This allows to apply modifications (e.g. decryption) while the mesh is still in an editable format. Applying the modifications to the mesh handle returned by oapiLoadMeshGlobal would not work in this case, because the mesh has already been copied to the client.

**15.53.2.9 OAPIFUNC const MESHHANDLE oapiLoadMeshGlobal (const char \**fname*)**

Retrieves a mesh handle from the global mesh manager.

When called for the first time for any given file name, the mesh is loaded from file and stored as a system resource. Every further request for the same mesh directly returns a handle to the stored mesh without additional file I/O.

**Parameters:**

*fname* mesh file name

**Returns:**

mesh handle

**Note:**

Once a mesh is globally loaded it remains in memory until the user closes the simulation window.

This function can be used to pre-load meshes to avoid load delays during the simulation. For example, parent objects may pre-load meshes for any child objects they may create later.

Do NOT delete any meshes obtained by this function with [oapiDeleteMesh\(\)](#) Orbiter takes care of deleting globally managed meshes.

If you assign the mesh to a vessel with a subsequent [VESSEL::AddMesh\(\)](#) call, a copy of the global mesh is created every time the vessel creates its visual, and discarded as soon as the visual is deleted. The global mesh can therefore be regarded as a template from which individual vessel instances make copies whenever they need to initialise their visual representation. Handles for the individual mesh copies can be obtained within the [VESSEL2::clbkVisualCreated\(\)](#) callback function, using the [VESSEL::GetMesh\(\)](#) method. Vessels should only modify their individual meshes, never the global template, since the latter is shared across all vessel instances.

For external graphics clients, the Orbiter core forwards the mesh data to the client for conversion to a device-specific format. The mesh template referred to by the handle returned by [oapiLoadMeshGlobal](#) is then no longer used, so any changes made to it will be ignored.

#### 15.53.2.10 OAPIFUNC SURFHANDLE oapiLoadTexture (const char \* *fname*, bool *dynamic* = false)

Load a texture from a file.

##### Parameters:

*fname* texture file name

*dynamic* allow dynamic modification

##### Returns:

Surface handle for the loaded texture, or NULL if not found.

##### Note:

Textures loaded by this function should be in DDS format and conform to the DirectX restrictions for texture surfaces, typically square bitmaps with dimensions of powers of 2 (128x128, 256x256, etc.). File names can contain search paths. Orbiter searches for textures in the standard way, i.e. first searches the HitexDir directory (usually Textures2), then the TextureDir directory (usually Textures). All search paths are relative to the texture root directories. For example, [oapiLoadTexture\(\)](#) ("myvessel\mytex.dds") would first search for Textures2\myvessel\mytex.dds, then for Textures\myvessel\mytex.dds.

#### 15.53.2.11 OAPIFUNC MESHGROUP\* oapiMeshGroup (MESHHANDLE *hMesh*, DWORD *idx*)

Returns a pointer to the group specification of a mesh group.

##### Parameters:

*hMesh* mesh handle

*idx* group index (>=0)

##### Returns:

pointer to mesh group specification (or NULL if idx out of range)

**Note:**

**MESHGROUP** is a structure that contains the components of the group, including vertex list, index list, texture and material index.

This method can be used to edit the a mesh group directly (for geometry animation, texture animation, etc.)

This function should only be applied to device-independent meshes, such as mesh templates.

For device-dependent mesh instances (such as returned by **VESSEL::GetDevMesh**) use **oapiEditMesh-Group** instead.

**See also:**

[oapiEditMeshGroup](#)

**15.53.2.12 OAPIFUNC DWORD oapiMeshGroupCount (MESHHANDLE *hMesh*)**

Returns the number of mesh groups defined in a mesh.

**Parameters:**

*hMesh* mesh handle

**Returns:**

number of mesh groups defined in the mesh

**Note:**

Each mesh is subdivided into mesh groups, defining a part of the 3-D object represented by the mesh. A group consists of a list of vertex coordinates and vertex indices, representing its geometry, and optionally a material and a texture reference.

See 3DModel document for details of the mesh format.

**15.53.2.13 OAPIFUNC MATERIAL\* oapiMeshMaterial (MESHHANDLE *hMesh*, DWORD *idx*)**

Returns a pointer to a material specification in the material list of the mesh.

**Parameters:**

*hMesh* mesh handle

*idx* material index ( $\geq 0$ )

**Returns:**

pointer to material specification (or NULL if idx out of range)

**Note:**

**MATERIAL** is a structure defined as follows:

```
typedef struct {    // material definition
    COLOUR4 diffuse; // diffuse component
    COLOUR4 ambient; // ambient component
    COLOUR4 specular; // specular component
    COLOUR4 emissive; // emissive component
    float power;     // specular power
} MATERIAL;
```

where `COLOUR4` defines a 4-valued (RGBA) colour component (red, green, blue, opacity):

```
typedef struct { // vertex definition including normals and texture coordinates
    float r; // red component
    float g; // green component
    float b; // blue component
    float a; // opacity
} COLOUR4;
```

colour component entries are in the range 0..1. Values > 1 may sometimes be used to obtain special effects.

This function can be used to edit mesh materials directly.

This function should only be used for mesh templates, not for device-specific rendering meshes (except for Orbiter's built-in graphics engine). For device meshes, use `oapiSetMaterial` instead.

#### See also:

[oapiAddMaterial](#), [oapiDeleteMaterial](#), [oapiMeshMaterialCount](#)

#### 15.53.2.14 OAPIFUNC DWORD oapiMeshMaterialCount (MESHHANDLE *hMesh*)

Returns the number of materials defined in the mesh.

##### Parameters:

*hMesh* mesh handle

##### Returns:

number of materials defined in the mesh

##### Note:

A mesh can contain a number of material specifications, and individual mesh groups can be linked to a material via the `MtrlIdx` entry in the group specification.

A material defines the diffuse, ambient, specular and emissive colour components of a mesh group, and also its level of transparency.

See 3DModel document for details of the mesh format.

#### 15.53.2.15 OAPIFUNC DWORD oapiMeshTextureCount (MESHHANDLE *hMesh*)

Returns the number of textures associated with a mesh.

##### Parameters:

*hMesh* mesh handle

##### Returns:

Number of textures

#### See also:

[oapiGetTextureHandle](#), [oapiSetTexture](#)

**15.53.2.16 OAPIFUNC VISHANDLE\* oapiObjectVisualPtr (OBJHANDLE *hObject*)**

Returns a pointer storing the objects visual handle.

**Parameters:**

*hObject* object handle

**Returns:**

pointer to visual handle

**Note:**

Returns a pointer that stores the object's visual handle whenever the object is within visual range of the camera. When the object is out of range, the pointer is set to NULL.

This function currently only works for vessel objects. All other object types return a pointer to NULL.

**15.53.2.17 OAPIFUNC void oapiParticleSetLevelRef (PSTREAM\_HANDLE *ph*, double \* *lvl*)**

Reset the reference pointer used by the particle stream to calculate the intensity (opacity) of the generated particles.

**Parameters:**

*ph* particle stream handle

*lvl* pointer to variable defining particle intensity

**Note:**

The variable pointed to by lvl should be set to values between 0 (lowest intensity) and 1 (highest intensity).

By default, exhaust streams are linked to the thrust level setting of the thruster they are associated with. Reentry streams are set to a fixed level of 1 by default.

This function allows to customise the appearance of the particle streams directly by the module.

Other parameters besides the intensity level, such as atmospheric density can also have an effect on the particle intensity.

**15.53.2.18 OAPIFUNC void oapiReleaseTexture (SURFHANDLE *hTex*)**

Release a texture.

**Parameters:**

*hTex* Texture surface handle.

**Note:**

After the function returns, the surface handle is invalid and should no longer be used.

Do not release textures that are referenced by a mesh. Mesh textures are released automatically.

**15.53.2.19 OAPIFUNC int oapiSetMaterial (DEVMESHHANDLE *hMesh*, DWORD *matidx*, const MATERIAL \* *mat*)**

Reset the properties of a mesh material.

**Parameters:**

*hMesh* device mesh handle  
*matidx* material index ( $\geq 0$ )  
*mat* pointer to new material properties.

**Returns:**

Error flag: 0=success, 1=no graphics engine attached, 2=graphics engine does not support operation, 3=invalid mesh handle, 4=material index out of range.

**Note:**

This function can be used to reset the parameters of an existing mesh material.

To add a new material, use [oapiAddMaterial](#) instead.

**15.53.2.20 OAPIFUNC bool oapiSetMeshProperty (DEVMESHHANDLE *hMesh*, DWORD *property*, DWORD *value*)**

Set custom properties for a device-specific mesh.

**Parameters:**

*hMesh* device mesh handle  
*property* property tag  
*value* new mesh property value

**Returns:**

*true* if the property tag was recognised and the request could be executed, *false* otherwise.

**Note:**

Currently only a single mesh property is recognised, but this may be extended in future versions:

- MESHPROPERTY\_MODULATEMATALPHA
- if *value*==0 (default) disable material alpha information in textured mesh groups (only use texture alpha channel).  
 if *value*<>0 modulate (mix) material alpha values with texture alpha maps.

**See also:**

[oapiSetMeshProperty\(MESHHANDLE,DWORD,DWORD\)](#)

**15.53.2.21 OAPIFUNC bool oapiSetMeshProperty (MESHHANDLE *hMesh*, DWORD *property*, DWORD *value*)**

Set custom properties for a mesh.

**Parameters:**

*hMesh* mesh handle  
*property* property tag  
*value* new mesh property value

**Returns:**

*true* if the property tag was recognised and the request could be executed, *false* otherwise.

**Note:**

Currently only a single mesh property is recognised, but this may be extended in future versions:

- MESHPROPERTY\_MODULATEMATERIALPHA  
if value==0 (default) disable material alpha information in textured mesh groups (only use texture alpha channel).  
if value<>0 modulate (mix) material alpha values with texture alpha maps.

**See also:**

[oapiSetMeshProperty\(DEVMESSHHANDLE,DWORD,DWORD\)](#)

**15.53.2.22 OAPIFUNC bool oapiSetTexture (MESSHHANDLE *hMesh*, DWORD *texidx*, SURFHANDLE *tex*)**

Replace a mesh texture.

**Parameters:**

*hMesh* mesh handle  
*texidx* texture index (>=1)  
*tex* texture handle

**Returns:**

*true* if texture was set successfully, *false* if texidx is out of range.

**Note:**

This function replaces one of the mesh textures. All mesh groups referencing the corresponding texture index will show the new texture.

texidx must be in the range [1..n] where n is the length of the texture list in the mesh, i.e. textures can be replaced, but no new textures added.

To point an individual mesh group to a different texture, use [oapiMeshGroup\(\)](#) to retrieve a [MESHGROUP](#) pointer, and modify the TexIdx entry.

## 15.54 HUD, MFD and panel functions

### Functions

- OAPIFUNC bool [oapiSetHUDMode](#) (int mode)  
*Set HUD (head up display) mode.*

- OAPIFUNC bool `oapiSetHUDMode` (int mode, const **HUDPARAM** \*prm)  
*Set HUD (head up display) mode with mode-specific parameters.*
- OAPIFUNC int `oapiGetHUDMode` ()  
*Query current HUD (head up display) mode.*
- OAPIFUNC int `oapiGetHUDMode` (**HUDPARAM** \*prm)  
*Query current HUD mode and mode parameters.*
- OAPIFUNC void `oapiToggleHUColour` ()  
*Switch the HUD display to a different colour.*
- OAPIFUNC void `oapiIncHUDIntensity` ()  
*Increase the brightness of the HUD display.*
- OAPIFUNC void `oapiDecHUDIntensity` ()  
*Decrease the brightness of the HUD display.*
- OAPIFUNC void `oapiRenderHUD` (**MESHHANDLE** hMesh, **SURFHANDLE** \*hTex)  
*Render custom HUD elements.*
- OAPIFUNC void `oapiOpenMFD` (int mode, int mfd)  
*Set an **MFD** (multifunctional display) to a specific mode.*
- OAPIFUNC void `oapiToggleMFD_on` (int mfd)  
*Switches an **MFD** on or off.*
- OAPIFUNC int `oapiGetMFDMode` (int mfd)  
*Get the current mode of the specified **MFD**.*
- OAPIFUNC int `oapiBroadcastMFDMessage` (int mode, int msg, void \*data)
- OAPIFUNC int `oapiSendMFDKey` (int mfd, DWORD key)  
*Sends a keystroke to an **MFD**.*
- OAPIFUNC void `oapiRefreshMFDButtons` (int mfd, **OBJHANDLE** hVessel=0)  
*Sends a `clbkMFDMode` call to the current focus vessel to allow it to dynamically update its button labels.*
- OAPIFUNC bool `oapiProcessMFDButton` (int mfd, int bt, int event)  
*Requests a default action as a result of a **MFD** button event.*
- OAPIFUNC const char \* `oapiMFDButtonLabel` (int mfd, int bt)  
*Retrieves a default label for an **MFD** button.*
- OAPIFUNC void `oapiRegisterMFD` (int mfd, const **MFDSPEC** &spec)  
*Registers an **MFD** position for a custom panel.*
- OAPIFUNC void `oapiRegisterMFD` (int mfd, const **EXTMFDSPEC** \*spec)  
*Registers an **MFD** position for a custom panel or virtual cockpit. This version has an extended parameter list.*

- OAPIFUNC void **oapiRegisterExternMFD** (**ExternMFD** \*emfd, const MFDSPEC &spec)
- OAPIFUNC bool **oapiUnregisterExternMFD** (**ExternMFD** \*emfd)
- OAPIFUNC void **oapiRegisterPanelBackground** (HBITMAP hBmp, DWORD flag=PANEL\_ATTACH\_BOTTOM|PANEL\_MOVEOUT\_BOTTOM, DWORD ck=(DWORD)-1)
 

*Register the background bitmap for a custom panel.*
- OAPIFUNC void **oapiRegisterPanelArea** (int id, const RECT &pos, int draw\_event=PANEL\_REDRAW\_NEVER, int mouse\_event=PANEL\_MOUSE\_IGNORE, int bkmode=PANEL\_MAP\_NONE)
 

*Defines a rectangular area within a panel to receive mouse or redraw notifications.*
- OAPIFUNC void **oapiSetPanelNeighbours** (int left, int right, int top, int bottom)
 

*Defines the neighbour panels of the current panels. These are the panels the user can switch to via Ctrl-Arrow keys.*
- OAPIFUNC void **oapiTriggerPanelRedrawArea** (int panel\_id, int area\_id)
 

*Triggers a redraw notification for a panel area.*
- OAPIFUNC void **oapiTriggerRedrawArea** (int panel\_id, int vc\_id, int area\_id)
 

*Triggers a redraw notification to either a 2D panel or a virtual cockpit.*
- OAPIFUNC bool **oapiBltPanelAreaBackground** (int area\_id, SURFHANDLE surf)
 

*Copies the stored background of a panel area into the provided surface.*
- OAPIFUNC void **oapiSetDefNavDisplay** (int mode)
 

*Defines how the navigation mode buttons will be displayed in a default cockpit view.*
- OAPIFUNC void **oapiSetDefRCSDisplay** (int mode)
 

*Enable or disable the display of the reaction control system indicators/controls in default cockpit view.*
- OAPIFUNC int **oapiSwitchPanel** (int direction)
 

*Switch to a neighbour instrument panel in 2-D panel cockpit mode.*
- OAPIFUNC int **oapiSetPanel** (int panel\_id)
 

*Switch to a different instrument panel in 2-D panel cockpit mode.*
- OAPIFUNC void **oapiSetPanelBlink** (**VECTOR3** v[4])

### 15.54.1 Function Documentation

#### 15.54.1.1 OAPIFUNC bool **oapiBltPanelAreaBackground** (int *area\_id*, SURFHANDLE *surf*)

Copies the stored background of a panel area into the provided surface.

This function should only be called from within the repaint callback function of an area registered with the PANEL\_MAP\_BGONREQUEST flag.

##### Parameters:

- area\_id*** area identifier
- surf*** surface handle

**Note:**

Areas defined with the PANEL\_MAP\_BGONREQUEST receive a surface with undefined contents when their repaint callback is called. They can use oapiBltPanelAreaBackground to copy the area background into the surface.

For areas not registered with the PANEL\_MAP\_BGONREQUEST, this function will do nothing.

Using PANEL\_MAP\_BGONREQUEST is more efficient than PANEL\_MAP\_BACKGROUND if the area doesn't need to be repainted at each call of the callback function, because it delays blitting the background until the module requests the background. This is particularly significant for areas which are updated at each time step.

**See also:**

[oapiRegisterPanelArea](#), [oapiRegisterPanelBackground](#)

**15.54.1.2 OAPIFUNC void oapiDecHUDIntensity ()**

Decrease the brightness of the HUD display.

**Note:**

Calling this function will decrease the intensity (in virtual cockpit modes) or brightness (in other modes) of the HUD display down to a minimum value.

This function should be called repeatedly (e.g. while the user presses a key).

**15.54.1.3 OAPIFUNC int oapiGetHUDMode (HUDPARAM \* *prm*)**

Query current HUD mode and mode parameters.

**Parameters:**

*prm* pointer to HUD parameter structure to be filled.

**Returns:**

Current HUD mode

**See also:**

[HUD Modes](#), [HUDPARAM](#), [oapiGetHUDMode\(\)](#)

**15.54.1.4 OAPIFUNC int oapiGetHUDMode ()**

Query current HUD (head up display) mode.

**Returns:**

Current HUD mode

**See also:**

[HUD Modes](#), [oapiGetHUDMode\(const HUDPARAM\\*\)](#), [oapiSetHUDMode](#)

**15.54.1.5 OAPIFUNC int oapiGetMFDMode (int *mfd*)**

Get the current mode of the specified [MFD](#).

**Parameters:**

*mfd* [MFD](#) identifier (e.g. MFD\_LEFT, MFD\_RIGHT)

**Returns:**

[MFD Mode](#)

**See also:**

[MFD Identifiers](#)

**15.54.1.6 OAPIFUNC void oapiIncHUDIntensity ()**

Increase the brightness of the [HUD](#) display.

**Note:**

Calling this function will increase the intensity (in virtual cockpit modes) or brightness (in other modes) of the [HUD](#) display up to a maximum value.

This function should be called repeatedly (e.g. while the user presses a key).

**See also:**

[oapiToggleHudColour](#)

**15.54.1.7 OAPIFUNC const char\* oapiMFDButtonLabel (int *mfd*, int *bt*)**

Retrieves a default label for an [MFD](#) button.

**Parameters:**

*mfd* [MFD](#) identifier (e.g. MFD\_LEFT, MFD\_RIGHT)

*bt* button number ( $\geq 0$ )

**Returns:**

pointer to static string containing the label, or NULL if the button is not assigned.

**Note:**

Labels contain 1 to 3 characters.

This function can be used to paint the labels on the [MFD](#) buttons of a custom panel.

The labels correspond to the default button actions executed by `VESSEL::ProcessMFDButton()`.

**See also:**

[MFD Identifiers](#)

**15.54.1.8 OAPIFUNC void oapiOpenMFD (int *mode*, int *mfd*)**

Set an **MFD** (multifunctional display) to a specific mode.

**Parameters:**

*mode* **MFD** mode

*mfd* **MFD** identifier (e.g. MFD\_LEFT, MFD\_RIGHT)

**Note:**

*mode* MFD\_NONE will turn off the **MFD**.

For the on-screen instruments, only MFD\_LEFT and MFD\_RIGHT are supported. Custom panels may support (up to 3) additional MFDs.

**See also:**

[MFD Identifiers](#), [MFD Modes](#)

**15.54.1.9 OAPIFUNC bool oapiProcessMFDButton (int *mfd*, int *bt*, int *event*)**

Requests a default action as a result of a **MFD** button event.

**Parameters:**

*mfd* **MFD** identifier (e.g. MFD\_LEFT, MFD\_RIGHT)

*bt* button number ( $\geq 0$ )

*event* mouse event (a combination of **PANEL\_MOUSE\_XXX** flags)

**Returns:**

Returns *true* if the button was processed, *false* if no action was assigned to the button.

**Note:**

Orbiter assigns default button actions for the various **MFD** modes. For example, in Orbit mode the action assigned to button 0 is Select reference. Calling oapiProcessMFDButton (for example as a reaction to a mouse button event) will execute this action.

**15.54.1.10 OAPIFUNC void oapiRefreshMFDButtons (int *mfd*, **OBJHANDLE** *hVessel* = 0)**

Sends a clbkMFDMode call to the current focus vessel to allow it to dynamically update its button labels.

**Parameters:**

*mfd* **MFD** identifier (e.g. MFD\_LEFT, MFD\_RIGHT)

*hVessel* recipient vessel handle

**Note:**

This message will only be sent to the current input focus vessel. If *hVessel*  $\neq 0$ , the function will not have any effect unless *hVessel* points to the focus vessel.

The recipient vessel will receive a **VESSEL2::clbkMFDMode** call, with the mode parameter set to MFD\_REFRESHBUTTONS.

This function can be used to force an [MFD](#) to refresh its button labels even if the mode has not changed. This is useful to update the labels for modes that dynamically update their labels. You don't need to call `oapiRefreshMFDButtons` after an actual mode change, because a `clbkMFD-Mode` call will be sent automatically by Orbiter.

#### See also:

[MFD Identifiers](#)

#### 15.54.1.11 OAPIFUNC void `oapiRegisterMFD` (*int mfd*, *const EXTMFDSPEC \* spec*)

Registers an [MFD](#) position for a custom panel or virtual cockpit. This version has an extended parameter list.

#### Parameters:

*mfd* [MFD](#) identifier (e.g. `MFD_LEFT`, `MFD_RIGHT`)

*spec* extended [MFD](#) parameters (see below)

#### Note:

Should be called in the body of [VESSEL2::clbkLoadPanel\(\)](#) or [VESSEL2::clbkLoadVC\(\)](#) to define [MFD](#) instruments for 2-D instrument panels or 3-D virtual cockpits.

`EXTMFDSPEC` is a structure with the following interface:

```
typedef struct {
    RECT pos;          // position of MFD in panel (pixel)
    DWORD nmesh;       // mesh index (>=0)
    DWORD ngroup;      // mesh group index (>=0)
    DWORD flag;        // parameter flags (see below)
    int nbt1;          // number of buttons in array 1 (e.g. left side of MFD display)
    int nbt2;          // number of buttons in array 2 (e.g. right side of MFD display)
    int bt_yofs;        // y-offset of top button from top display edge (pixel)
    int bt_ydist;       // y-distance between buttons (pixel)
} MFDSPEC;
```

*flag* is a bitmask which can be set to a combination of the following options:

- `MFD_SHOWMODELABELS` Show 3-letter abbreviations for [MFD](#) modes when displaying the mode selection page (default: only show carets ">"). This is useful if the buttons are not located next to the list display.

If this function is used during initialisation of a 2-D instrument panel, *pos* defines the rectangle of the [MFD](#) display in the panel bitmap (in pixels), while *nmesh* and *ngroup* are ignored.

If it is used during initialisation of a virtual cockpit, *nmesh* and *ngroup* define the mesh and group index of the mesh element which will receive the [MFD](#) display texture, while *pos* is ignored.

#### See also:

[MFD Identifiers](#)

#### 15.54.1.12 OAPIFUNC void `oapiRegisterMFD` (*int mfd*, *const MFDSPEC & spec*)

Registers an [MFD](#) position for a custom panel.

#### Parameters:

*mfd* [MFD](#) identifier (e.g. `MFD_LEFT`, `MFD_RIGHT`)

*spec* MFD parameters (see below)

**Note:**

Should be called in the body of [VESSEL2::clbkLoadPanel\(\)](#) for panels which define MFDs.  
Defining more than 2 or 3 MFDs per panel can degrade performance.  
MFDSPEC is a structure with the following interface:

```
typedef struct {
    RECT pos;           // position of MFD in panel (pixel)
    int nbt_left;      // number of buttons on left side of MFD display
    int nbt_right;     // number of buttons on right side of MFD display
    int bt_yofs;       // y-offset of top button from top display edge (pixel)
    int bt_ydist;      // y-distance between buttons (pixel)
} MFDSPEC;
```

**See also:**

[MFD Identifiers](#)

### 15.54.1.13 OAPIFUNC void oapiRegisterPanelArea (int *id*, const RECT & *pos*, int *draw\_event* = PANEL\_REDRAW\_NEVER, int *mouse\_event* = PANEL\_MOUSE\_IGNORE, int *bkmode* = PANEL\_MAP\_NONE)

Defines a rectangular area within a panel to receive mouse or redraw notifications.

**Parameters:**

- id* area identifier
- pos* bounding box of the marked area
- draw\_event* defines redraw events
- mouse\_event* defines mouse events
- bkmode* redraw background mode

**Note:**

Each panel area must be defined with an identifier aid which is unique within the panel.  
*draw\_event* can have the following values:

- PANEL\_REDRAW\_NEVER : do not generate redraw events.
- PANEL\_REDRAW\_ALWAYS : generate a redraw event at every time step.
- PANEL\_REDRAW\_MOUSE : mouse events trigger redraw events.

For possible values of *mouse\_event* see [Mouse event identifiers](#). *PANEL\_MOUSE\_IGNORE* prevents mouse events from being triggered.

By default, no mouse events are sent during a playback session. You can force Orbiter to trigger mouse events during a playback (e.g. to allow the user to operate [MFD](#) buttons) by using *PANEL\_MOUSE\_ONREPLAY* in combination with any of the other mouse event flags.

*bkmode* defines the bitmap handed to the redraw callback:

- PANEL\_MAP\_NONE : provides an undefined bitmap. Should be used if the whole area is repainted.
- PANEL\_MAP\_CURRENT : provides a copy of the current area.
- PANEL\_MAP\_BACKGROUND : provides a copy of the panel background (as defined by [oapiRegisterPanelBackground\(\)](#)).

- **PANEL\_MAP\_BGONREQUEST**: like **PANEL\_MAP\_BACKGROUND**, this stores the area background, but the user must request it explicitly with a call to `oapiBltPanelAreaBackground`. This can improve performance if the area does not need to be updated at each call of the repaint callback function.

**See also:**

[Mouse event identifiers](#), [Panel redraw events](#), [oapiRegisterPanelBackground](#)

**15.54.1.14 OAPIFUNC void oapiRegisterPanelBackground (HBITMAP *hBmp*, DWORD *flag* = PANEL\_ATTACH\_BOTTOM|PANEL\_MOVEOUT\_BOTTOM, DWORD *ck* = (DWORD)-1)**

Register the background bitmap for a custom panel.

**Parameters:**

- hBmp*** bitmap handle
- flag*** property bit flags (see notes)
- ck*** transparency colour key

**Note:**

This function will normally be called in the body of `ovcLoadPanel`.

Typically the bitmap will be stored as a resource in the DLL and obtained by a call to the Windows function `LoadBitmap(...)`.

*flag* defines panel properties and can be a combination of the following bitmask:

- **PANEL\_ATTACH\_{LEFT/RIGHT/TOP/BOTTOM}**
- **PANEL\_MOVEOUT\_{LEFT/RIGHT/TOP/BOTTOM}**

where **PANEL\_ATTACH\_BOTTOM** means that the bottom edge of the panel cannot be scrolled above the bottom edge of the screen (other directions work equivalently) and **PANEL\_MOVEOUT\_BOTTOM** means that the panel can be scrolled downwards out of the screen (other directions work equivalently). The colour key, if defined, specifies a colour which will appear transparent when displaying the panel. The key is in (hex) 0xRRGGBB format. If no key is specified, the panel will be opaque. It is best to use black (0x000000) or white (0xffffffff) as colour keys, since other values may cause problems in 16bit screen modes. Of course, care must be taken that the keyed colour does not appear anywhere in the opaque part of the panel.

**See also:**

[oapiRegisterPanelArea](#)

**15.54.1.15 OAPIFUNC void oapiRenderHUD (MESHHANDLE *hMesh*, SURFHANDLE \* *hTex*)**

Render custom HUD elements.

**Parameters:**

- hMesh*** HUD mesh handle
- hTex*** array of texture handles

**Note:**

This function should only be called from within [VESSEL3::clbkRenderHUD](#).  
It can be used to render custom HUD elements in glass cockpit and 2-D panel mode.  
The mesh handle must refer to a 2-D mesh (z-components of all vertices are zero). The x and y components are in units of screen pixels.  
The mesh may have multiple groups, but generally a single group should be sufficient. The texture indices of each group refer to the textures in the hTex list (starting with 0). If only a single texture is used, the texture index in the mesh should be set to 0, and hTex should be a pointer to the surface handle.  
Mesh animations can be applied by modifying vertex and/or texture coordinates at each frame.

**15.54.1.16 OAPIFUNC int oapiSendMFDKey (int *mfld*, DWORD *key*)**

Sends a keystroke to an [MFD](#).

**Parameters:**

*mfld* [MFD](#) identifier (e.g. MFD\_LEFT, MFD\_RIGHT)  
*key* key code (see [OAPI\\_KEY\\_xxx Constants](#))

**Returns:**

nonzero if the [MFD](#) understood and processed the key.

**Note:**

This function can be used to interact with the [MFD](#) as if the user had pressed Shift-key, for example to select a different [MFD](#) mode, to select a target body, etc.

**See also:**

[MFD Identifiers](#)

**15.54.1.17 OAPIFUNC void oapiSetDefNavDisplay (int *mode*)**

Defines how the navigation mode buttons will be displayed in a default cockpit view.

**Parameters:**

*mode* display mode (0 .. 2)

**Note:**

This function should usually be called in the body of the overloaded [VESSEL2::clbkLoadGenericCockpit\(\)](#).

It defines if the buttons for navigation modes (e.g. "Killrot" or "Prograde") are displayed in the generic (non-panel) cockpit camera mode, and if the buttons can be operated with the mouse.

The following values for mode are defined:

- 0 buttons are not shown
- 1 buttons are shown and can be operated with the mouse (default)
- 2 only buttons representing active modes are shown, and can not be operated with the mouse

**15.54.1.18 OAPIFUNC void oapiSetDefRCSDisplay (int mode)**

Enable or disable the display of the reaction control system indicators/controls in default cockpit view.

**Parameters:**

*mode* display mode (0 .. 1)

**Note:**

This function should usually be called in the body of the overloaded [VES-SEL2::clbkLoadGenericCockpit\(\)](#).

The RCS display consists of three buttons in the engine status display at the top left of the generic cockpit view. If displayed (mode=1), the buttons show the RCS mode (off/rotational/linear), and can be clicked with the mouse to switch modes.

The following values for mode are defined:

- 0 RCS buttons are not shown
- 1 RCS buttons are shown and can be operated with the mouse (default)

**15.54.1.19 OAPIFUNC bool oapiSetHUDMode (int mode, const HUDPARAM \*prm)**

Set HUD (head up display) mode with mode-specific parameters.

**Parameters:**

*mode* new HUD mode

*prm* mode-specific parameters

**Returns:**

*true* if mode has changed, *false* otherwise.

**Note:**

Mode `HUD_NONE` will turn off the HUD display.

See constants `HUD_xxx` for currently supported HUD modes.

**See also:**

[HUD Modes](#), [HUDPARAM](#), [oapiGetHUDMode](#), [oapiGetHUDMode\(const HUDPARAM\\*\)](#)

**15.54.1.20 OAPIFUNC bool oapiSetHUDMode (int mode)**

Set HUD (head up display) mode.

**Parameters:**

*mode* new HUD mode

**Returns:**

*true* if mode has changed, *false* otherwise.

**Note:**

Mode `HUD_NONE` will turn off the HUD display.

See constants `HUD_xxx` for currently supported HUD modes.

**See also:**

[HUD Modes](#), [oapiGetHUDMode](#)

**15.54.1.21 OAPIFUNC int oapiSetPanel (int *panel\_id*)**

Switch to a different instrument panel in 2-D panel cockpit mode.

**Parameters:**

*panel\_id* panel identifier ( $>=0$ )

**Returns:**

panel\_id if the panel was set successfully, or -1 if failed (camera not in 2-D panel cockpit mode, or requested panel does not exist for the current vessel)

**Note:**

This function has no effect if the current view is not in 2-D panel cockpit mode.

**See also:**

[oapiSwitchPanel](#)

**15.54.1.22 OAPIFUNC void oapiSetPanelNeighbours (int *left*, int *right*, int *top*, int *bottom*)**

Defines the neighbour panels of the current panels. These are the panels the user can switch to via Ctrl-Arrow keys.

**Parameters:**

*left* panel id of left neighbour (or -1 if none)

*right* panel id of right neighbour (or -1 if none)

*top* panel id of top neighbour (or -1 if none)

*bottom* panel id of bottom neighbour (or -1 if none)

**Note:**

This function should be called during panel registration (in [VESSEL2::clbkLoadPanel\(\)](#)) to define the neighbours of the registered panel.

Every panel (except panel 0) must be listed as a neighbour by at least one other panel, otherwise it is inaccessible.

**15.54.1.23 OAPIFUNC int oapiSwitchPanel (int *direction*)**

Switch to a neighbour instrument panel in 2-D panel cockpit mode.

**Parameters:**

*direction* neighbour direction (see notes)

**Returns:**

Identifier of the newly selected panel ( $>=0$ ) or -1 if the requested panel does not exist.

**Note:**

direction can be one of the constants in [Panel neighbour identifiers](#).

The neighbourhood status between panels is established by the [oapiSetPanelNeighbours\(\)](#) function.

This function has no effect if the current view is not in 2-D panel cockpit mode.

**See also:**

[oapiSetPanel](#)

**15.54.1.24 OAPIFUNC void oapiToggleHUDColour ()**

Switch the HUD display to a different colour.

**Note:**

Orbiter currently defines 3 HUD colours: green, red, white. Calls to [oapiToggleHUDColour](#) will cycle through these.

**See also:**

[oapiIncHUDIntensity](#), [oapiDecHUDIntensity](#)

**15.54.1.25 OAPIFUNC void oapiToggleMFD\_on (int *mfd*)**

Switches an [MFD](#) on or off.

**Parameters:**

*mfd* MFD identifier (e.g. MFD\_LEFT, MFD\_RIGHT)

**Note:**

Flips the on/off state of an [MFD](#). Typically used to respond to the user pressing the "power" button.

**See also:**

[oapiOpenMFD](#)

**15.54.1.26 OAPIFUNC void oapiTriggerPanelRedrawArea (int *panel\_id*, int *area\_id*)**

Triggers a redraw notification for a panel area.

**Parameters:**

*panel\_id* panel identifier ( $\geq 0$ )

*area\_id* area identifier ( $\geq 0$ )

**Note:**

The redraw notification is ignored if the requested panel is not currently displayed.

**See also:**

[oapiTriggerRedrawArea](#)

**15.54.1.27 OAPIFUNC void oapiTriggerRedrawArea (int panel\_id, int vc\_id, int area\_id)**

Triggers a redraw notification to either a 2D panel or a virtual cockpit.

**Parameters:**

- panel\_id* identifier for the panel to receive the redraw message
- vc\_id* identifier for the virtual cockpit to receive the redraw message
- area\_id* area identifier

**Note:**

This function can be used to combine the functionality of the [oapiTriggerPanelRedrawArea\(\)](#) and [oapiVCTriggerRedrawArea\(\)](#) methods. Depending on the current cockpit mode, Orbiter sends the redraw request to either ovcPanelRedrawEvent() or ovcVCRedrawEvent().

This method can only be used if the panel and virtual cockpit areas share a common area identifier.

## 15.55 Drawing support functions

### Enumerations

- enum **FontStyle** { **FONT\_NORMAL** = 0, **FONT\_BOLD** = 1, **FONT\_ITALIC** = 2, **FONT\_UNDERLINE** = 4 }

### Functions

- OAPIFUNC [oapi::Sketchpad](#) \* [oapiGetSketchpad](#) ([SURFHANDLE](#) surf)  
*Obtain a drawing context for a surface.*
- OAPIFUNC void [oapiReleaseSketchpad](#) ([oapi::Sketchpad](#) \*skp)  
*Release a drawing device context instance.*
- OAPIFUNC [oapi::Font](#) \* [oapiCreateFont](#) (int height, bool prop, char \*face, FontStyle style=**FONT\_NORMAL**)  
*Creates a font resource for drawing text into surfaces.*
- OAPIFUNC [oapi::Font](#) \* [oapiCreateFont](#) (int height, bool prop, const char \*face, FontStyle style, int orientation)  
*Creates a font resource for drawing text into surfaces.*
- OAPIFUNC void [oapiReleaseFont](#) ([oapi::Font](#) \*font)  
*Release a font resource.*
- OAPIFUNC [oapi::Pen](#) \* [oapiCreatePen](#) (int style, int width, [DWORD](#) col)  
*Creates a pen resource for drawing lines and shape outlines.*
- OAPIFUNC void [oapiReleasePen](#) ([oapi::Pen](#) \*pen)  
*Release a pen resource.*
- OAPIFUNC [oapi::Brush](#) \* [oapiCreateBrush](#) ([DWORD](#) col)  
*Creates a brush resource for filling shapes.*

- OAPIFUNC void [oapiReleaseBrush](#) ([oapi::Brush](#) \*brush)  
*Release a brush resource.*
- OAPIFUNC HDC [oapiGetDC](#) ([SURFHANDLE](#) surf)  
*Obtain a Windows device context handle (HDC) for a surface.*
- OAPIFUNC void [oapiReleaseDC](#) ([SURFHANDLE](#) surf, HDC hDC)  
*Release a GDI drawing device context handle.*

### 15.55.1 Function Documentation

#### 15.55.1.1 OAPIFUNC [oapi::Brush\\*](#) [oapiCreateBrush](#) (DWORD col)

Creates a brush resource for filling shapes.

##### Parameters:

*col* shape fill colour (format: 0xBBGGRR)

##### Note:

After use, the brush should be deallocated with [oapiReleaseBrush](#).

##### See also:

[oapiReleaseBrush](#)

#### 15.55.1.2 OAPIFUNC [oapi::Font\\*](#) [oapiCreateFont](#) (int height, bool prop, const char \* face, FontStyle style, int orientation)

Creates a font resource for drawing text into surfaces.

##### Parameters:

*height* font height [pixel]  
*prop* flag for proportional/fixed pitch font  
*face* typeface name (see notes)  
*style* font decoration style (see notes)  
*orientation* text orientation [1/10 deg]

##### Returns:

pointer to font resource, or NULL if not supported.

##### Note:

Identical to [oapiCreateFont\(int,bool,char\\*,FontStyle\)](#), but contains the additional orientation parameter.

**15.55.1.3 OAPIFUNC oapi::Font\* oapiCreateFont (int *height*, bool *prop*, char \**face*, FontStyle *style* = FONT\_NORMAL)**

Creates a font resource for drawing text into surfaces.

**Parameters:**

- height* font height [pixel]
- prop* flag for proportional/fixed pitch font
- face* typeface name (see notes)
- style* font decoration style (see notes)

**Returns:**

pointer to font resource, or NULL if not supported.

**Note:**

The following generic typeface names should be understood by all graphics systems:

- Fixed (fixed pitch font)
- Sans (sans-serif proportional font)
- Serif (serif proportional font) Other font names may not be recognised by all graphics clients. In that case, the default fixed or sans-serif font will be used, depending on the value of *prop*.

The decoration style flags allow bold, italic and underlining.

After use, the font should be deallocated with oapiReleaseFont.

**See also:**

[oapiReleaseFont](#)

**15.55.1.4 OAPIFUNC oapi::Pen\* oapiCreatePen (int *style*, int *width*, DWORD *col*)**

Creates a pen resource for drawing lines and shape outlines.

**Parameters:**

- style* line style (0=invisible, 1=solid, 2=dashed)
- width* line width [pixel]
- col* line colour (format: 0xBBGGRR)

**Note:**

After use, the pen should be deallocated with oapiReleasePen.

**See also:**

[oapiReleasePen](#)

**15.55.1.5 OAPIFUNC HDC oapiGetDC (SURFHANDLE *surf*)**

Obtain a Windows device context handle (HDC) for a surface.

**Parameters:**

*surf* surface handle

**Returns:**

device context handle, or NULL if not supported.

**Warning:**

This function uses a device-dependent drawing context handle and may not work with all graphics clients. It has been superseded by oapiGetSketchpad.

**Note:**

This function returns a valid device handle only when Orbiter is using its inline graphics client, or if an external client is attached that supports GDI drawing. In all other cases, the function returns NULL. Therefore, the caller should always check the returned value before using it.

If a nonzero HDC was returned, it should be released with [oapiReleaseDC](#) after drawing.

Most graphics clients must lock the surface data buffer (and copy it to main memory, if necessary) before GDI access can be provided. This means that read/write access to the surface (e.g. for blitting) may be disabled between oapiGetDC and oapiReleaseDC, and should be avoided.

**See also:**

[oapiReleaseDC](#), [oapiGetSketchpad](#)

**15.55.1.6 OAPIFUNC oapi::Sketchpad\* oapiGetSketchpad (SURFHANDLE *surf*)**

Obtain a drawing context for a surface.

**Parameters:**

*surf* surface handle

**Returns:**

drawing context instance, or NULL if no graphics support

**Note:**

This function returns a valid context instance only when Orbiter is attached to a graphics client which supports 2-D drawing into surfaces. The caller should check the return value for NULL.

If a nonzero Sketchpad instance was returned, it should be released with [oapiReleaseSketchpad](#) after drawing.

Most graphics clients must lock the surface data buffer (and copy it to main memory, if necessary) before drawing access can be provided. This means that read/write access to the surface (e.g. for blitting) may be disabled between oapiGetSketchpad and oapiReleaseSketchpad, and should be avoided.

**See also:**

[oapiReleaseSketchpad](#)

**15.55.1.7 OAPIFUNC void oapiReleaseBrush (oapi::Brush \* *brush*)**

Release a brush resource.

**Parameters:**

*brush* pointer to brush resource

**See also:**

[oapiCreateBrush](#)

**15.55.1.8 OAPIFUNC void oapiReleaseDC (SURFHANDLE *surf*, HDC *hDC*)**

Release a GDI drawing device context handle.

**Parameters:**

*surf* surface handle

*hDC* device context handle

**Warning:**

This function uses a device-dependent drawing context handle and may not work with all graphics clients. It has been superseded by oapiReleaseSketchpad.

**Note:**

Use this function to release a device context previously acquired with [oapiGetDC](#).

Standard Windows device context rules apply. For example, any custom device objects loaded via SelectObject must be unloaded before calling oapiReleaseDC.

**See also:**

[oapiGetDC](#), [oapiGetSketchpad](#), [oapiReleaseSketchpad](#)

**15.55.1.9 OAPIFUNC void oapiReleaseFont (oapi::Font \* *font*)**

Release a font resource.

**Parameters:**

*font* pointer to font resource

**See also:**

[oapiCreateFont](#)

**15.55.1.10 OAPIFUNC void oapiReleasePen (oapi::Pen \* *pen*)**

Release a pen resource.

**Parameters:**

*pen* pointer to pen resource

**See also:**

[oapiCreatePen](#)

**15.55.1.11 OAPIFUNC void oapiReleaseSketchpad (oapi::Sketchpad \* skp)**

Release a drawing device context instance.

**Parameters:**

*skp* drawing context instance

**Note:**

Use this function to release a device instance previously acquired with oapiGetSketchpad.

**See also:**

[oapiGetSketchpad](#)

## 15.56 Surface functions

**Functions**

- OAPIFUNC SURFHANDLE [oapiCreateSurface](#) (int width, int height)  
*Create a surface of the specified dimensions.*
- OAPIFUNC SURFHANDLE [oapiCreateSurface](#) (HBITMAP hBmp, bool release\_bmp=true)  
*Create a surface from a bitmap. Bitmap surfaces are typically used for blitting operations during instrument panel redraws.*
- OAPIFUNC SURFHANDLE [oapiCreateTextureSurface](#) (int width, int height)  
*Create a surface that can be used as a texture for a 3-D object.*
- OAPIFUNC void [oapiDestroySurface](#) (SURFHANDLE surf)  
*Destroy a surface previously created with oapiCreateSurface.*
- OAPIFUNC void [oapiClearSurface](#) (SURFHANDLE surf, DWORD col=0)
- OAPIFUNC void [oapiSetSurfaceColourKey](#) (SURFHANDLE surf, DWORD ck)  
*Define a colour key for a surface to allow transparent blitting.*
- OAPIFUNC void [oapiClearSurfaceColourKey](#) (SURFHANDLE surf)  
*Clear a previously defined colour key.*
- OAPIFUNC void [oapiBlt](#) (SURFHANDLE tgt, SURFHANDLE src, int tgtx, int tgty, int srcx, int srcy, int w, int h, DWORD ck=SURF\_NO\_CK)  
*Copy a rectangular area from one surface to another.*
- OAPIFUNC void [oapiBlt](#) (SURFHANDLE tgt, SURFHANDLE src, RECT \*tgtr, RECT \*srcr, DWORD ck=SURF\_NO\_CK, DWORD rotate=SURF\_NO\_ROTATION)  
*Copy a scaled rectangular area from one surface to another.*
- OAPIFUNC int [oapiBeginBltGroup](#) (SURFHANDLE tgt)  
*Begin a block of blitting operations to the same target surface.*
- OAPIFUNC int [oapiEndBltGroup](#) ()

*End a block of blitting operations to the same target surface.*

- OAPIFUNC void [oapiColourFill](#) (SURFHANDLE *tgt*, DWORD *fillcolor*, int *tgtx*=0, int *tgy*=0, int *w*=0, int *h*=0)

*Fill an area of the target surface with a uniform colour.*

### 15.56.1 Function Documentation

#### 15.56.1.1 OAPIFUNC int [oapiBeginBltGroup](#) (SURFHANDLE *tgt*)

Begin a block of blitting operations to the same target surface.

##### Parameters:

*tgt* Target surface for subsequent blitting calls.

##### Returns:

Error code:

- 0: success
- -1: no graphics client loaded
- -1: blitting target was set already
- -2: *tgt* was given as RENDERTGT\_NONE
- other error codes are client-specific

##### Note:

All blitting calls following this function must address the same target until the blitting block is ended with a call to [oapiEndBltGroup](#).

This mechanism should always be used when multiple blitting operations go to the same target, because some graphics clients may be able to optimise the calls.

To refer to the main render surface, use *tgt*=RENDERTGT\_MAINWINDOW.

Do not call a blitting function with a target other than *tgt* within a blitting group.

Within the blitting group, multiple source surfaces can be used, but if possible the blitting calls should be arranged so that blits from the same surface are grouped together, to allow additional optimisation.

##### See also:

[oapiEndBltGroup](#)

#### 15.56.1.2 OAPIFUNC void [oapiBlt](#) (SURFHANDLE *tgt*, SURFHANDLE *src*, RECT \**tgtr*, RECT \**srcr*, DWORD *ck* = SURF\_NO\_CK, DWORD *rotate* = SURF\_NO\_ROTATION)

Copy a scaled rectangular area from one surface to another.

##### Parameters:

*tgt* target surface

*src* source surface

*tgtr* pointer to target rectangle [pixel]

*srcr* pointer to source rectangle [pixel]

*ck* transparency colour key (inline graphics only)

*rotate* rotation flag (deprecated)

**Note:**

This function copies a rectangular area from a source to a target surface.

If the sizes of the source and target rectangles differ, the copied area is stretched or shrunk to fit into the target rectangle.

This function must not be used while a device context is acquired for the target surface (i.e. between [oapiGetDC](#) and [oapiReleaseDC](#) calls).

Transparent blitting can be performed by specifying a colour key in *ck*. The transparent colour can either be passed explicitly in *ck*, or *ck* can be set to SURF\_PREDEF\_CK to use the key previously defined with [oapiSetSurfaceColourKey](#).

Colour keys are only supported with Orbiter's inline graphics client. External clients ignore the *ck* parameter. The use of colour keys is therefore discouraged.

The rotation flag is deprecated. It has no effect.

### 15.56.1.3 OAPIFUNC void oapiBlt (SURFHANDLE *tgt*, SURFHANDLE *src*, int *tgtx*, int *tgyt*, int *srx*, int *sry*, int *w*, int *h*, DWORD *ck* = SURF\_NO\_CK)

Copy a rectangular area from one surface to another.

**Parameters:**

*tgt* target surface

*src* source surface

*tgtx* left edge of target rectangle [pixel]

*tgyt* top edge of target rectangle [pixel]

*srx* left edge of source rectangle [pixel]

*sry* top edge of source rectangle [pixel]

*w* width of copied rectangle [pixel]

*h* height of copied rectangle [pixel]

*ck* transparency colour key (inline graphics only)

**Note:**

This function copies rectangular areas between two surfaces, or between two locations of the same surface.

A typical use is the dynamic update of instrument panels, e.g. in the body of [VES-SEL2::clbkPanelRedrawEvent](#).

This function must not be used while a device context is acquired for the target surface (i.e. between [oapiGetDC](#) and [oapiReleaseDC](#) calls). If a blitting operation is necessary between [oapiGetDC](#) and [oapiReleaseDC](#), you must use the standard Windows BitBlt function. However this does not use hardware acceleration and should therefore be avoided.

Transparent blitting can be performed by specifying a colour key in *ck*. The transparent colour can either be passed explicitly in *ck*, or *ck* can be set to SURF\_PREDEF\_CK to use the key previously defined with [oapiSetSurfaceColourKey](#).

Colour keys are only supported with Orbiter's inline graphics client. External clients ignore the *ck* parameter. The use of colour keys is therefore discouraged.

**15.56.1.4 OAPIFUNC void oapiClearSurfaceColourKey (SURFHANDLE *surf*)**

Clear a previously defined colour key.

**Parameters:**

*surf* surface handle

**See also:**

[oapiSetSurfaceColourKey](#), [oapiBlt](#)

**15.56.1.5 OAPIFUNC void oapiColourFill (SURFHANDLE *tgt*, DWORD *fillcolor*, int *tgtx* = 0, int *tgyt* = 0, int *w* = 0, int *h* = 0)**

Fill an area of the target surface with a uniform colour.

**Parameters:**

*tgt* target surface

*fillcolor* fill colour

*tgtx* coordinate of upper left corner of area to fill.

*tgyt* coordinate of upper left corner of area to fill.

*w* width of area to fill.

*h* height of area to fill.

**Note:**

The fill colour should be acquired with [oapiGetColour\(\)](#), to ensure compatibility with 16-bit colour modes.

This function must not be used while a device context is acquired for the target surface (i.e. between [oapiGetDC\(\)](#) and [oapiReleaseDC\(\)](#) calls).

If w and h are zero (the default) the whole surface is filled. The tgtx and tgyt values are ignored in that case and can be omitted.

**15.56.1.6 OAPIFUNC SURFHANDLE oapiCreateSurface (HBITMAP *hBmp*, bool *release\_bmp* = true)**

Create a surface from a bitmap. Bitmap surfaces are typically used for blitting operations during instrument panel redraws.

**Parameters:**

*hBmp* bitmap handle

*release\_bmp* flag for bitmap release

**Returns:**

Handle to the new surface.

**Note:**

The easiest way to access bitmaps is by storing them as resources in the module, and loading them via a call to LoadBitmap.

Do not use this function with a bitmap generated by CreateBitmap. To create a surface of specified dimensions, use oapiCreateSurface (width, height) instead.

If *release\_bmp==true*, then [oapiCreateSurface\(\)](#) will destroy the bitmap after creating a surface from it (i.e. the hBmp handle will be invalid after the function returns), otherwise the module is responsible for destroying the bitmap by a call to DestroyObject when it is no longer needed.

Surfaces should be destroyed by calling oapiDestroySurface when they are no longer needed.

**See also:**

[oapiDestroySurface](#)

#### 15.56.1.7 OAPIFUNC SURFHANDLE [oapiCreateSurface \(int width, int height\)](#)

Create a surface of the specified dimensions.

**Parameters:**

*width* width of surface bitmap (pixels)

*height* height of surface bitmap (pixels)

**Returns:**

Handle to the new surface.

**Note:**

The bitmap contents are undefined after creation, so the surface must be repainted fully before mapping it to the screen.

If you want to use the surface as a texture, use oapiCreateTextureSurface instead.

Surfaces should be destroyed by calling oapiDestroySurface when they are no longer needed.

**See also:**

[oapiDestroySurface](#)

#### 15.56.1.8 OAPIFUNC SURFHANDLE [oapiCreateTextureSurface \(int width, int height\)](#)

Create a surface that can be used as a texture for a 3-D object.

**Parameters:**

*width* width of surface bitmap (pixels)

*height* height of surface bitmap (pixels)

**Returns:**

handle of new texture surface

**Note:**

Use this function instead of oapiCreateSurface if you want the surface to be used as a surface texture for a 3-D object, for example via a call to oapiSetTexture.

For maximum compatibility, the surface should be square, and dimensions powers of 2, for example 64x64, 128x128, 256x256, etc. Note that older video cards may not support textures larger than 256x256.

Surfaces should be destroyed by calling oapiDestroySurface when they are no longer needed.

**15.56.1.9 OAPIFUNC void oapiDestroySurface (SURFHANDLE *surf*)**

Destroy a surface previously created with oapiCreateSurface.

**Parameters:**

*surf* surface handle

**15.56.1.10 OAPIFUNC int oapiEndBltGroup ()**

End a block of blitting operations to the same target surface.

**Returns:**

Error code:

- 0; success
- -1: no graphics client loaded
- -2: no blitting target was set
- other error codes are client-specific

**See also:**

[oapiBeginBltGroup](#)

**15.56.1.11 OAPIFUNC void oapiSetSurfaceColourKey (SURFHANDLE *surf*, DWORD *ck*)**

Define a colour key for a surface to allow transparent blitting.

**Parameters:**

*surf* surface handle

*ck* colour key (0xRRGGBB)

**Note:**

Defining a colour key and subsequently calling oapiBlt with the SURF\_PREDEF\_CK flag is slightly more efficient than passing the colour key explicitly to oapiBlt each time, if the same colour key is used repeatedly.

**See also:**

[oapiClearSurfaceColourKey](#), [oapiBlt](#)

## 15.57 Custom MFD mode definition

### Functions

- OAPIFUNC int [oapiRegisterMFDMode](#) (MFDMODESPECEX &spec)  
*Register a custom MFD mode.*
- OAPIFUNC bool [oapiUnregisterMFDMode](#) (int mode)  
*Unregister a previously registered custom MFD mode.*

- OAPIFUNC void [oapiDisableMFDMode](#) (int mode)  
*Disable an MFD mode.*
- OAPIFUNC int [oapiGetMFDModeSpecEx](#) (char \*name, MFDMODESPECEX \*\*spec=0)  
*Returns the mode identifier and spec for an MFD mode defined by its name.*

### 15.57.1 Function Documentation

#### 15.57.1.1 OAPIFUNC void oapiDisableMFDMode (int mode)

Disable an MFD mode.

##### Parameters:

*mode* MFD mode to be disabled.

##### Note:

The list of disabled MFDs is cleared whenever the focus switches to a new vessel. To disable MFD modes permanently for a particular vessel type, [oapiDisableMFDMode\(\)](#) should be called from within the `ovcFocusChanged()` callback function.

For builtin MFD modes, mode can be any of the MFD\_xxx constants. For MFD modes defined in plugin modules, the mode id must be obtained by a call to [oapiGetMFDModeSpec\(\)](#).

##### See also:

[MFD Modes](#)

#### 15.57.1.2 OAPIFUNC int oapiGetMFDModeSpecEx (char \* name, MFDMODESPECEX \*\* spec = 0)

Returns the mode identifier and spec for an MFD mode defined by its name.

##### Parameters:

*name* MFD name (as defined in MFDMODESPECEX::name during [oapiRegisterMFDMode\(\)](#))

*spec* If defined, this will return a pointer to the MFDMODESPECEX structure for the mode.

##### Returns:

MFD mode identifier.

##### Note:

This function returns the same value as [oapiRegisterMFDMode\(\)](#) for the given mode.

If no matching mode is found, the return value is MFD\_NONE. In that case, the returned spec pointer is undefined.

The mode identifiers for custom MFD modes can not be assumed to persist across simulation runs, since they will change if the user loads or unloads MFD plugins.

This function can also be used for built-in MFD modes, which are defined as follows:

Name string	Mode identifier
Orbit	MFD_ORBIT
Surface	MFD_SURFACE
Map	MFD_MAP
HSI	MFD_HSI
VOR/VTOL	MFD_LANDING
Docking	MFD_DOCKING
Align Planes	MFD_OPLANEALIGN
Sync Orbit	MFD_OSYNC
Transfer	MFD_TRANSFER
COM/NAV	MFD_COMMS

### 15.57.1.3 OAPIFUNC int oapiRegisterMFDMode (MFDMODESPECEX & *spec*)

Register a custom [MFD](#) mode.

**Parameters:**

*spec* [MFD](#) specs (see notes below)

**Returns:**

[MFD](#) mode identifier

**Note:**

This function registers a custom [MFD](#) mode with Orbiter. There are two types of custom MFDs: generic and vessel class-specific. Generic [MFD](#) modes are available to all vessel types, while specific modes are only available for a single vessel class. Generic modes should be registered in the [InitModule](#) callback function of a plugin module. Vessel class specific modes are not implemented yet.

[MFDMODESPECEX](#) is a struct defining the parameters of the new mode:

```
typedef struct {
    char *name;      // points to the name of the new mode
    DWORD key;       // mode selection key
    void *context;   // mode-specific context pointer
    int (*msgproc)(UINT,UINT,WPARAM,LPARAM); // address of MFD message parser
} MFDMODESPEC;
```

See `orbitersdk\samples\CustomMFD` for a sample [MFD](#) mode implementation.

**See also:**

[oapiUnregisterMFDMode](#)

### 15.57.1.4 OAPIFUNC bool oapiUnregisterMFDMode (int *mode*)

Unregister a previously registered custom [MFD](#) mode.

**Parameters:**

*mode* mode identifier, as returned by [oapiRegisterMFDMode](#)

**Returns:**

*true* on success (mode could be unregistered).

## 15.58 Virtual cockpit functions

### Functions

- OAPIFUNC void [oapiVCRegisterMFD](#) (int mfd, const VCMFDSPEC \*spec)  
*Define a render target for rendering an **MFD** display in a virtual cockpit.*
- OAPIFUNC void [oapiVCRegisterArea](#) (int id, const RECT &tgtrect, int draw\_event, int mouse\_event, int bkmode, SURFHANDLE tgt)  
*Define an active area in a virtual cockpit. Active areas can be repainted. This function is similar to [oapiRegisterPanelArea](#).*
- OAPIFUNC void [oapiVCRegisterArea](#) (int id, int draw\_event, int mouse\_event)  
*Define an active area in a virtual cockpit. This version is used when no dynamic texture update is required during redraw events.*
- OAPIFUNC void [oapiVCSetAreaClickmode\\_Spherical](#) (int id, const VECTOR3 &cnt, double rad)  
*Associate a spherical region in the virtual cockpit with a registered area to receive mouse events.*
- OAPIFUNC void [oapiVCSetAreaClickmode\\_Quadrilateral](#) (int id, const VECTOR3 &p1, const VECTOR3 &p2, const VECTOR3 &p3, const VECTOR3 &p4)  
*Associate a quadrilateral region in the virtual cockpit with a registered area to receive mouse events.*
- OAPIFUNC void [oapiVCSetNeighbours](#) (int left, int right, int top, int bottom)  
*Defines the neighbouring virtual cockpit camera positions in relation to the current position. The user can switch to neighbour positions with Ctrl-Arrow keys.*
- OAPIFUNC void [oapiVCTriggerRedrawArea](#) (int vc\_id, int area\_id)  
*Triggers a redraw notification for a virtual cockpit area.*
- OAPIFUNC void [oapiVCRegisterHUD](#) (const VCHUDSPEC \*spec)  
*Define a render target for the head-up display (HUD) in a virtual cockpit.*

### 15.58.1 Function Documentation

#### 15.58.1.1 OAPIFUNC void [oapiVCRegisterArea](#) (int *id*, int *draw\_event*, int *mouse\_event*)

Define an active area in a virtual cockpit. This version is used when no dynamic texture update is required during redraw events.

##### Parameters:

- id* area identifier  
*draw\_event* redraw condition (see [draw events](#))  
*mouse\_event* mouse event (see [mouse events](#))

##### Note:

This function is equivalent to:

```
oapiVCRegisterArea (aid, _R(0,0,0,0), draw_event, mouse_event, PANEL_MAP_NONE, NULL);
```

### 15.58.1.2 OAPIFUNC void oapiVCRegisterArea (int *id*, const RECT & *tgtrect*, int *draw\_event*, int *mouse\_event*, int *bkmode*, SURFHANDLE *tgt*)

Define an active area in a virtual cockpit. Active areas can be repainted. This function is similar to oapiRegisterPanelArea.

**Parameters:**

- id* area identifier
- tgtrect* bounding box of the active area in the target texture (pixels)
- draw\_event* redraw condition (see [draw events](#))
- mouse\_event* mouse event ( see [mouse events](#))
- bkmode* background mode (see [bkmodes](#))
- tgt* target texture to be updated

**Note:**

The target texture can be retrieved from a mesh by using the [oapiGetTextureHandle\(\)](#) method. Dynamic textures must be marked with flag "D" in the mesh file.

Redraw events can be used not only to update mesh textures dynamically, but also to animate mesh groups, or edit mesh vertices or texture coordinates.

If no dynamic texture repaints are required during redraw events, use the alternative version of [oapiVCRegisterArea\(\)](#) instead.

To define a mouse-sensitive volume in the virtual cockpit, use one of the *oapiVCSetAreaClickmode\_-XXX* functions.

### 15.58.1.3 OAPIFUNC void oapiVCRegisterHUD (const VCHUDSPEC \* *spec*)

Define a render target for the head-up display (HUD) in a virtual cockpit.

**Parameters:**

- spec* hud specification (see notes)

**Note:**

This function should be placed in the body of the VESSEL2::ovcLoadVC() vessel module callback function.

VCHUDSPEC is a structure defined as:

```
struct VCHUDSPEC {
    DWORD nmesh;      // mesh index
    DWORD ngroup;     // group index
    VECTOR3 hudcnt;  // HUD centre in vessel frame
    double size;      // physical size of the HUD [m]
};
```

The mesh group specified by nmesh and ngroup should be a square panel in front of the camera position in the virtual cockpit. This group is rendered separately from the rest of the mesh and should therefore have FLAG 2 set in the mesh file. The group material and texture can be set to 0.

The HUD centre position and size are required to allow Orbiter to correctly scale the display.

Orbiter renders the HUD with completely transparent background. Rendering the glass pane, brackets, etc. is up to the vessel designer.

**15.58.1.4 OAPIFUNC void oapiVCRegisterMFD (int *mfd*, const VCMFDSPEC \* *spec*)**

Define a render target for rendering an [MFD](#) display in a virtual cockpit.

**Parameters:**

*mfd* [MFD](#) identifier (e.g. MFD\_LEFT, MFD\_RIGHT)  
*spec* render target specification (see notes)

**Note:**

The render target specification is defined as a structure:

```
struct VCMFDSPEC { DWORD nmesh, ngroup };
```

where nmesh is the mesh index ( $>=0$ ), and ngroup is the group index ( $>=0$ ) defining the render target. This function should be placed in the body of the ovcLoadVC vessel module callback function.

The addressed mesh group should define a simple square (4 vertices, 2 triangles). The group materials and textures can be set to 0.

**See also:**

[MFD Identifiers](#)

**15.58.1.5 OAPIFUNC void oapiVCSetAreaClickmode\_Quadrilateral (int *id*, const VECTOR3 & *p1*, const VECTOR3 & *p2*, const VECTOR3 & *p3*, const VECTOR3 & *p4*)**

Associate a quadrilateral region in the virtual cockpit with a registered area to receive mouse events.

**Parameters:**

*id* area identifier (as specified during area registration)  
*p1* top left corner of region  
*p2* top right corner  
*p3* bottom left corner  
*p4* bottom right corner

**Note:**

This function will trigger mouse events when the user clicks within the projection of the quadrilateral region on the render window. The mouse event handler will receive the relative position within the area at which the mouse event occurred, where the top left corner has coordinates (0,0), and the bottom right corner has coordinates (1,1).

The area can define any flat quadrilateral in space. It is not limited to rectangles, but all 4 points should be in the same plane.

**See also:**

[VESSEL2::clbkVCMouseEvent](#)

**15.58.1.6 OAPIFUNC void oapiVCSetAreaClickmode\_Spherical (int *id*, const VECTOR3 & *cnt*, double *rad*)**

Associate a spherical region in the virtual cockpit with a registered area to receive mouse events.

**Parameters:**

*id* area identifier (as specified during area registration)

*cnt* centre of active area in the local vessel frame

*rad* radius of active area [m]

**Note:**

The area identifier must refer to an area which has previously been registered with a call to [oapiVCRegisterArea\(\)](#), with the required mouse event modes.

This function can be called repeatedly, to change the mouse-sensitive area.

**See also:**

[VESSEL2::clbkVCMouseEvent](#)

**15.58.1.7 OAPIFUNC void oapiVCSetNeighbours (int *left*, int *right*, int *top*, int *bottom*)**

Defines the neighbouring virtual cockpit camera positions in relation to the current position. The user can switch to neighbour positions with Ctrl-Arrow keys.

**Parameters:**

*left* panel id of left neighbour position (or -1 if none)

*right* panel id of right neighbour position (or -1 if none)

*top* panel id of top neighbour position (or -1 if none)

*bottom* panel id of bottom neighbour position (or -1 if none)

**Note:**

This function should be called during virtual cockpit registration (in [VESSEL2::clbkLoadVC\(\)](#)) to define the neighbouring cockpit camera positions, if any.

The left, right, top and bottom values specify the (zero-based) identifiers of the VC positions to switch to when the user presses Ctrl and an arrow button, or -1 if no position is available in this direction.

The neighbour relations should normally be reciprocal, i.e. if position 0 defines position 1 as its right neighbour, then position 1 should define position 0 as its left neighbour.

If only a single VC position (id 0) is defined, this function doesn't need to be called.

Orbiter calls [VESSEL2::clbkLoadVC\(\)](#) with the appropriate id whenever the user switches to a new position.

**15.58.1.8 OAPIFUNC void oapiVCTriggerRedrawArea (int *vc\_id*, int *area\_id*)**

Triggers a redraw notification for a virtual cockpit area.

**Parameters:**

*vc\_id* virtual cockpit identifier

*area\_id* area identifier (as specified during area registration)

**Note:**

This function triggers a call to the VESSEL2::ovcVCRedrawEvent() callback function in the vessel module.

The redraw notification is normally only sent if vc\_id is equal to the currently active virtual cockpit position ( $>=0$ ). To invoke the redraw notification independent of the currently active position, set vc\_id to -1.

## 15.59 Customisation - custom menu, dialogs

### Typedefs

- `typedef void(* CustomFunc )(void *context)`

### Functions

- OAPIFUNC LAUNCHPADITEM\_HANDLE `oapiRegisterLaunchpadItem` (`LaunchpadItem *item`, `LAUNCHPADITEM_HANDLE parent=0`)  
*Register a new item in the parameter list of the "Extra" tab of the Orbiter Launchpad dialog.*
- OAPIFUNC bool `oapiUnregisterLaunchpadItem` (`LaunchpadItem *item`)  
*Unregister a previously registered entry in the "Extra" tab of the Orbiter Launchpad dialog.*
- OAPIFUNC LAUNCHPADITEM\_HANDLE `oapiFindLaunchpadItem` (`const char *name=0`, `LAUNCHPADITEM_HANDLE parent=0`)  
*Returns a handle for an existing entry in the Extra parameter list.*
- OAPIFUNC DWORD `oapiRegisterCustomCmd` (`char *label`, `char *desc`, `CustomFunc func`, `void *context`)  
*Register a custom function. Custom functions can be accessed in Orbiter by pressing Ctrl-F4. A common use for custom functions is opening plugin dialog boxes.*
- OAPIFUNC bool `oapiUnregisterCustomCmd` (`int cmdId`)  
*Unregister a previously defined custom function.*
- OAPIFUNC HWND `oapiOpenDialog` (`HINSTANCE hDLLInst`, `int resourceId`, `DLGPROC msgProc`, `void *context=0`)  
*Open a dialog box defined as a Windows resource.*
- OAPIFUNC HWND `oapiOpenDialogEx` (`HINSTANCE hDLLInst`, `int resourceId`, `DLGPROC msgProc`, `DWORD flag=0`, `void *context=0`)  
*Open a dialog box defined as a Windows resource. This version provides additional functionality compared to `oapiOpenDialog()`.*
- OAPIFUNC HWND `oapiFindDialog` (`HINSTANCE hDLLInst`, `int resourceId`)  
*Returns the window handle of an open dialog box, or NULL if the specified dialog box is not open.*
- OAPIFUNC void `oapiCloseDialog` (`HWND hDlg`)  
*Close a dialog box.*
- OAPIFUNC void \* `oapiGetDialogContext` (`HWND hDlg`)

*Retrieves the context pointer of a dialog box which has been defined during the call to [oapiOpenDialog\(\)](#).*

- OAPIFUNC bool **oapiRegisterWindow** (HINSTANCE hDLLInst, HWND hWnd, DWORD flag=0)
- OAPIFUNC bool [oapiAddTitleButton](#) (DWORD msgid, HBITMAP hBmp, DWORD flag)  
*Adds a custom button in the title bar of a dialog box.*
- OAPIFUNC DWORD [oapiGetTitleButtonState](#) (HWND hDlg, DWORD msgid)
- OAPIFUNC bool [oapiSetTitleButtonState](#) (HWND hDlg, DWORD msgid, DWORD state)
- OAPIFUNC BOOL [oapiDefDialogProc](#) (HWND hDlg, UINT uMsg, WPARAM wParam, LPARAM lParam)  
*Default Orbiter dialog message handler.*
- OAPIFUNC bool [oapiOpenHelp](#) (HELPCONTEXT \*hcontext)  
*Opens the ingame help window on the specified help page.*
- OAPIFUNC bool [oapiOpenLaunchpadHelp](#) (HELPCONTEXT \*hcontext)  
*Opens a help window outside a simulation session, i.e. when the Launchpad dialog is displayed.*

### 15.59.1 Function Documentation

#### 15.59.1.1 OAPIFUNC bool [oapiAddTitleButton](#) (DWORD *msgid*, HBITMAP *hBmp*, DWORD *flag*)

Adds a custom button in the title bar of a dialog box.

##### Parameters:

*msgid* The message identifier generated by pressing the button

*hBmp* bitmap containing the button images.

*flag* additional parameters (see notes)

##### Returns:

*true* if the button could be created, *false* otherwise.

##### Note:

`oapiAddTitleButton` can only be called while processing the `WM_INITDIALOG` message in the dialog message procedure.

Up to 5 buttons can be created in the title bar, including the standard buttons defined in the call to `oapiOpenDialogEx`.

Whenever the users left-clicks on the button, a `WM_COMMAND` message is generated in the message procedure, where the low-word of the `WPARAM` parameter is set to *msgid*.

The button size defined in the bitmap should be 15x15 pixels large. Their look should conform to Orbiter's standard dialog buttons.

The following bit-flags in the *flag* parameter are currently supported: `DLG_CB_TWOSTATE`: The button has two states, and clicking on it will flip between the two states.

If the `DLG_CB_TWOSTATE` flag is set, the bitmap must be 15x30 pixels large, containing two images, where the upper image represents the initial state, and the lower image represents the "checked" state. If the `DLG_CB_TWOSTATE` flag is set, the button state (0 or 1) is passed in the high-word of the `WPARAM` parameter whenever the dialog is notified of a button press.

**15.59.1.2 OAPIFUNC void oapiCloseDialog (HWND *hDlg*)**

Close a dialog box.

**Parameters:**

*hDlg* dialog window handle (as obtained by oapiOpenDialog)

**Note:**

This function should be called in response to an IDCANCEL message in the dialog message handler to close a dialog which was opened by [oapiOpenDialog\(\)](#).

**15.59.1.3 OAPIFUNC BOOL oapiDefDialogProc (HWND *hDlg*, UINT *uMsg*, WPARAM *wParam*, LPARAM *lParam*)**

Default Orbiter dialog message handler.

This function should be called from the message handler of all dialogs created with oapiOpenDialog to perform default actions for any messages not processed in the handler.

**Parameters:**

The parameters passed to the message handler.

**Returns:**

The value returned by oapiDefDialogProc should be returned by the message handler.

**Typical usage:**

```
BOOL CALLBACK MsgProc (HWND hDlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch (uMsg) {
        case WM_COMMAND:
            switch (LOWORD (wParam)) {
                case IDCANCEL: // dialog closed by user
                    CloseDlg (hDlg);
                    return TRUE;
                }
                break;
                // add more messages to be processed here
            }
            return oapiDefDialogProc (hDlg, uMsg, wParam, lParam);
}
```

**Note:**

oapiDefDialogProc currently only processes the WM\_SETCURSOR message, and always returns *false*.

**See also:**

[oapiCloseDialog](#), [oapiFindDialog](#), [oapiOpenDialog](#)

**15.59.1.4 OAPIFUNC HWND oapiFindDialog (HINSTANCE *hDLLInst*, int *resourceId*)**

Returns the window handle of an open dialog box, or NULL if the specified dialog box is not open.

**Parameters:**

*hDLLInst* module instance handle (as obtained from [InitModule](#))

*resourceId* dialog resource identifier

**Returns:**

Window handle of dialog box, or NULL if the dialog was not found.

**15.59.1.5 OAPIFUNC LAUNCHPADITEM\_HANDLE oapiFindLaunchpadItem (const char \* name = 0, LAUNCHPADITEM\_HANDLE parent = 0)**

Returns a handle for an existing entry in the Extra parameter list.

**Parameters:**

*name* the name of the item in the list (or 0 for first entry)

*parent* the parent item below which to search (or 0 for root)

**Returns:**

value Item handle if found, or 0 otherwise.

**Note:**

This method allows to retrieve the handle of an already existing entry in the Extra list. It is useful for placing new items below a parent that wasn't defined by the module itself.

It can be used iteratively to search for lower-level entries.

If name is not set, the first child entry of parent is returned (or the first root entry, if parent==0).

You should only attach children to items that don't themselves define an activation method.

**See also:**

[oapiRegisterLaunchpadItem](#), [oapiUnregisterLaunchpadItem](#)

**15.59.1.6 OAPIFUNC void\* oapiGetDialogContext (HWND *hDlg*)**

Retrieves the context pointer of a dialog box which has been defined during the call to [oapiOpenDialog\(\)](#).

**Parameters:**

*hDlg* dialog window handle

**Note:**

This function returns NULL if no context pointer was specified in [oapiOpenDialog\(\)](#).

**15.59.1.7 OAPIFUNC HWND oapiOpenDialog (HINSTANCE *hDLLInst*, int *resourceId*, DLGPROC *msgProc*, void \* *context* = 0)**

Open a dialog box defined as a Windows resource.

**Parameters:**

*hDLLInst* module instance handle (as obtained from [InitModule](#))

*resourceId* dialog resource identifier

*msgProc* pointer to Windows message handler

*context* optional user-defined pointer

**Returns:**

handle of the new dialog box, or NULL if the dialog was open already.

**Note:**

Use [oapiOpenDialog\(\)](#) instead of standard Windows methods such as CreateWindow or DialogBox, to make sure the dialog works in fullscreen mode.

Only one instance of a dialog box can be open at a time. A second call to [oapiOpenDialog\(\)](#) with the same dialog id will fail and return NULL.

The interface of the message handler is as follows:

```
BOOL CALLBACK MsgProc ( HWND hDlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
```

See standard Windows documentation for usage of the dialog message handler.

The context pointer can be set to user-defined data which can be retrieved via the [oapiGetDialogContext\(\)](#) function. This allows to pass data into the message handler.

Note that [oapiGetDialogContext\(\)](#) can not be used when processing the WM\_INITDIALOG message. In this case, the context pointer can be accessed via lParam instead.

**See also:**

[oapiFindDialog](#), [oapiCloseDialog](#), [oapiOpenDialogEx](#)

### 15.59.1.8 OAPIFUNC HWND oapiOpenDialogEx (HINSTANCE *hDLLInst*, int *resourceId*, DLGPROC *msgProc*, DWORD *flag* = 0, void \* *context* = 0)

Open a dialog box defined as a Windows resource. This version provides additional functionality compared to [oapiOpenDialog\(\)](#).

**Parameters:**

*hDLLInst* module instance handle (as obtained from `InitModule`)

*resourceId* dialog resource identifier

*msgProc* pointer to Windows message handler

*flag* bit-flags to define dialog box options (see notes)

*context* optional user-defined pointer

**Returns:**

handle of the new dialog box, or NULL if the box could not be opened.

**Note:**

The flag parameter can be a combination of the following values:

- **DLG\_ALLOWMULTI**: Allows multiple instances of the same dialog resource to be open simultaneously.
- **DLG\_CAPTIONCLOSE**: Shows a Close button in the dialog title bar. Pressing it produces an IDCANCEL notification to the message procedure.
- **DLG\_CAPTIONHELP**: Shows a Help button in the dialog title bar. Pressing it produces an IDHELP notification to the message procedure.

If customised title bar buttons are requested, the dialog box template should not contain standard title buttons, by omitting the `WS_SYSMENU` window style.

Additional buttons can be created by using the `oapiAddTitleButton` function.

**See also:**

[oapiFindDialog](#), [oapiCloseDialog](#), [oapiGetDialogContext](#)

#### 15.59.1.9 OAPIFUNC bool oapiOpenHelp (HELPCONTEXT \* *hcontext*)

Opens the ingame help window on the specified help page.

**Parameters:**

*hcontext* help context structure.

**Returns:**

Currently always returns *true*.

#### 15.59.1.10 OAPIFUNC bool oapiOpenLaunchpadHelp (HELPCONTEXT \* *hcontext*)

Opens a help window outside a simulation session, i.e. when the Launchpad dialog is displayed.

**Parameters:**

*hcontext* help context structure.

**Returns:**

Currently always returns *true*.

#### 15.59.1.11 OAPIFUNC DWORD oapiRegisterCustomCmd (char \* *label*, char \* *desc*, CustomFunc func, void \* *context*)

Register a custom function. Custom functions can be accessed in Orbiter by pressing Ctrl-F4. A common use for custom functions is opening plugin dialog boxes.

**Parameters:**

*label* label to appear in the custom function list.

*desc* a short description of the function

*func* pointer to the function to be executed

*context* pointer to custom data which will be passed to func

**Returns:**

function identifier

**Note:**

The interface of the custom function is defined as follows:

```
typedef void (*CustomFunc) (void *context)
```

where context is the pointer passed to [oapiRegisterCustomCmd\(\)](#).

**See also:**

[oapiUnregisterCustomCmd](#)

### 15.59.1.12 OAPIFUNC LAUNCHPADITEM\_HANDLE oapiRegisterLaunchpadItem (LaunchpadItem \* *item*, LAUNCHPADITEM\_HANDLE *parent* = 0)

Register a new item in the parameter list of the "Extra" tab of the Orbiter Launchpad dialog.

**Parameters:**

*item* pointer to [LaunchpadItem](#) structure (see notes)

*parent* parent item, or NULL for root item

**Returns:**

Handle for the new item

**Note:**

The "Extra" list of the Launchpad dialog is customisable and can be used by modules to allow user selection of global parameters and settings. Data can be written to/read from file and therefore persist across Orbiter sessions.

Item is a pointer to a class instance derived from [LaunchpadItem](#). It defines what is displayed in the list, and how the user accesses the item.

Items can be arranged in a hierarchy. Child items can be defined by passing the handle of a previous item as the parent parameter.

If an entry with the same name as item->Name() already exists, no new entry is generated, and the handle of the existing entry is returned.

Because double-clicking on an item both activates it and expands the child list of parent items, parent items should be inert (i.e. should not define their clbkOpen method) to avoid ambiguities.

[oapiRegisterLaunchpadItem\(\)](#) should usually be called during the DLL initialisation function. A matching [oapiUnregisterLaunchpadItem\(\)](#) should be called during the DLL exit function.

**See also:**

[oapiUnregisterLaunchpadItem](#), [oapiFindLaunchpadItem](#)

### 15.59.1.13 OAPIFUNC bool oapiUnregisterCustomCmd (int *cmdId*)

Unregister a previously defined custom function.

**Parameters:**

*cmdId* custom function identifier (as returned by [oapiRegisterCustomCmd\(\)](#))

**Returns:**

*false* indicates failure (cmdId not recognised)

**See also:**

[oapiRegisterCustomCmd](#)

**15.59.1.14 OAPIFUNC bool oapiUnregisterLaunchpadItem (LaunchpadItem \* item)**

Unregister a previously registered entry in the "Extra" tab of the Orbiter Launchpad dialog.

**Parameters:**

*item* handle of the item to be removed

**Returns:**

value *true* if item could be unregistered, *false* if no matching item was found.

**Note:**

A module must unregister all the launchpad items it has registered before it is unloaded, at the latest during ExitModule. Failing to do so will leave stale items in the parameter list of the Extra tab, leading to undefined behaviour.

**See also:**

[oapiRegisterLaunchpadItem](#), [oapiFindLaunchpadItem](#)

## 15.60 File IO Functions

### Functions

- OAPIFUNC FILEHANDLE [oapiOpenFile](#) (const char \*fname, FileAccessMode mode, PathRoot root=ROOT)  
*Open a file for reading or writing.*
- OAPIFUNC void [oapiCloseFile](#) (FILEHANDLE file, FileAccessMode mode)  
*Close a file after reading or writing.*
- OAPIFUNC bool [oapiSaveScenario](#) (const char \*fname, const char \*desc)  
*Writes the current simulation state to a scenario file.*
- OAPIFUNC void [oapiWriteLine](#) (FILEHANDLE file, char \*line)  
*Writes a line to a file.*
- OAPIFUNC void [oapiWriteLog](#) (char \*line)  
*Writes a line to the Orbiter log file (*orbiter.log*) in the main orbiter directory.*
- OAPIFUNC void [oapiWriteLogV](#) (const char \*format,...)  
*Writes a formatted string with variable number of arguments to *orbiter.log*.*
- OAPIFUNC void [oapiWriteScenario\\_string](#) (FILEHANDLE scn, char \*item, char \*string)  
*Writes a string-valued item to a scenario file.*
- OAPIFUNC void [oapiWriteScenario\\_int](#) (FILEHANDLE scn, char \*item, int i)  
*Writes an integer-valued item to a scenario file.*
- OAPIFUNC void [oapiWriteScenario\\_float](#) (FILEHANDLE scn, char \*item, double d)  
*Writes a floating point-valued item to a scenario file.*

- OAPIFUNC void [oapiWriteScenario\\_vec](#) (FILEHANDLE scn, char \*item, const VECTOR3 &vec)  
*Writes a vector-valued item to a scenario file.*
- OAPIFUNC bool [oapiReadScenario\\_nextline](#) (FILEHANDLE scn, char \*&line)  
*Reads an item from a scenario file.*
- OAPIFUNC bool [oapiReadItem\\_string](#) (FILEHANDLE f, char \*item, char \*string)  
*Read the value of a tag from a configuration file.*
- OAPIFUNC bool [oapiReadItem\\_float](#) (FILEHANDLE f, char \*item, double &d)  
*Read the value of a tag from a configuration file.*
- OAPIFUNC bool [oapiReadItem\\_int](#) (FILEHANDLE f, char \*item, int &i)  
*Read the value of a tag from a configuration file.*
- OAPIFUNC bool [oapiReadItem\\_bool](#) (FILEHANDLE f, char \*item, bool &b)  
*Read the value of a tag from a configuration file.*
- OAPIFUNC bool [oapiReadItem\\_vec](#) (FILEHANDLE f, char \*item, VECTOR3 &vec)  
*Read the value of a tag from a configuration file.*
- OAPIFUNC void [oapiWriteItem\\_string](#) (FILEHANDLE f, char \*item, char \*string)  
*Write a tag and its value to a configuration file.*
- OAPIFUNC void [oapiWriteItem\\_float](#) (FILEHANDLE f, char \*item, double d)  
*Write a tag and its value to a configuration file.*
- OAPIFUNC void [oapiWriteItem\\_int](#) (FILEHANDLE f, char \*item, int i)  
*Write a tag and its value to a configuration file.*
- OAPIFUNC void [oapiWriteItem\\_bool](#) (FILEHANDLE f, char \*item, bool b)  
*Write a tag and its value to a configuration file.*
- OAPIFUNC void [oapiWriteItem\\_vec](#) (FILEHANDLE f, char \*item, const VECTOR3 &vec)  
*Write a tag and its value to a configuration file.*

### 15.60.1 Function Documentation

#### 15.60.1.1 OAPIFUNC void [oapiCloseFile](#) (FILEHANDLE *file*, FileAccessMode *mode*)

Close a file after reading or writing.

##### Parameters:

- file* file handle  
*mode* access mode with which the file was opened

**Note:**

Use this function on files opened with oapiOpenFile after finishing with it.  
The file access mode passed to oapiCloseFile must be the same as used to open it.

**15.60.1.2 OAPIFUNC FILEHANDLE oapiOpenFile (const char \**fname*, FileAccessMode *mode*, PathRoot *root* = ROOT)**

Open a file for reading or writing.

**Parameters:**

- fname* file name (with optional path)
- mode* read/write mode (see notes)
- root* path origin (see notes)

**Returns:**

file handle

**Note:**

The following access modes are supported:

- FILE\_IN read
- FILE\_OUT write (overwrite)
- FILE\_APP write (append)

The file path defined in fname is relative to either the main Orbiter folder or to one of Orbiter's default subfolders, depending on the root parameter:

- ROOT Orbiter main directory
- CONFIG Orbiter config folder
- SCENARIOS Orbiter scenarios folder
- TEXTURES Orbiter standard texture folder
- TEXTURES2 Orbiter high-res texture folder
- MESHES Orbiter mesh folder
- MODULES Orbiter module folder

You should always specify a standard Orbiter subfolder by the above mechanism, rather than manually as a path in fname, because Orbiter installations can redirect these directories.

Be careful when opening a file for writing in the standard Orbiter subfolders: except for ROOT and SCENARIOS, all other standard folders may be readonly (e.g. for CD installations)

**See also:**

[oapiCloseFile](#)

**15.60.1.3 OAPIFUNC bool oapiReadItem\_bool (FILEHANDLE *f*, char \* *item*, bool & *b*)**

Read the value of a tag from a configuration file.

**Parameters:**

*f* file handle  
*item* tag defining the item  
*b* boolean value

**Returns:**

*true* if tag was found in the file, *false* if not.

**Note:**

In a file boolean values are represented by the strings "FALSE" and "TRUE".

**See also:**

[oapiReadItem\\_string](#) for more details.

**15.60.1.4 OAPIFUNC bool oapiReadItem\_float (FILEHANDLE *f*, char \* *item*, double & *d*)**

Read the value of a tag from a configuration file.

**Parameters:**

*f* file handle  
*item* tag defining the item  
*d* double value

**Returns:**

*true* if tag was found in the file, *false* if not.

**See also:**

[oapiReadItem\\_string](#) for more details.

**15.60.1.5 OAPIFUNC bool oapiReadItem\_int (FILEHANDLE *f*, char \* *item*, int & *i*)**

Read the value of a tag from a configuration file.

**Parameters:**

*f* file handle  
*item* tag defining the item  
*i* integer value

**Returns:**

*true* if tag was found in the file, *false* if not.

**See also:**

[oapiReadItem\\_string](#) for more details.

**15.60.1.6 OAPIFUNC bool oapiReadItem\_string (FILEHANDLE *f*, char \* *item*, char \* *string*)**

Read the value of a tag from a configuration file.

**Parameters:**

*f* file handle  
*item* tag defining the item  
*string* character-string value

**Returns:**

*true* if tag was found in the file, *false* if not.

**Note:**

The tag-value entries of a configuration file have the format <tag> = <value>  
The functions search the complete file independent of the current position of the file pointer.  
Whitespace around tag and value are discarded, as well as comments beginning with a semicolon (;) to the end of the line.  
String values can contain internal whitespace.

**15.60.1.7 OAPIFUNC bool oapiReadItem\_vec (FILEHANDLE *f*, char \* *item*, VECTOR3 & *vec*)**

Read the value of a tag from a configuration file.

**Parameters:**

*f* file handle  
*item* tag defining the item  
*vec* vector value

**Returns:**

*true* if tag was found in the file, *false* if not.

**Note:**

Vector values are represented by space-separated triplets of floating point values.

**See also:**

[oapiReadItem\\_string](#) for more details.

**15.60.1.8 OAPIFUNC bool oapiReadScenario\_nextline (FILEHANDLE *scn*, char \*& *line*)**

Reads an item from a scenario file.

**Parameters:**

*scn* file handle  
*line* pointer to the scanned line

**Note:**

The function returns *true* as long as an item for the current block could be read. It returns false at EOF, or when an "END" token is read.

Leading and trailing whitespace, and trailing comments (from ";" to EOL) are automatically removed. "line" points to an internal static character buffer. The buffer grows automatically to hold lines of arbitrary length.

The buffer is overwritten on the next call to oapiReadScenario\_nextline, so it must be copied or processed before the next call.

**Examples:**

[clbkLoadStateEx.cpp](#).

**15.60.1.9 OAPIFUNC bool oapiSaveScenario (const char \**fname*, const char \**desc*)**

Writes the current simulation state to a scenario file.

**Parameters:**

*fname* scenario file name

*desc* scenario description

**Returns:**

*true* if scenario could be written successfully, *false* if an error occurred.

**Note:**

The file name is always calculated relative from the default orbiter scenario folder (usually Orbiter\Scenarios). The file name can contain a relative path starting from that directory, but the subdirectories must already exist. The function will not create new directories. The file name should not contain an absolute path.

The file name should not contain an extension. Orbiter will automatically add a .scn extension.

The description string can be empty ("").

**15.60.1.10 OAPIFUNC void oapiWriteItem\_bool (FILEHANDLE *f*, char \**item*, bool *b*)**

Write a tag and its value to a configuration file.

**Parameters:**

*f* file handle

*item* pointer to tag string

*b* boolean value

**Note:**

In a file boolean values are represented by the strings "FALSE" and "TRUE".

**See also:**

[oapiWriteItem\\_string](#) for more details

**15.60.1.11 OAPIFUNC void oapiWriteItem\_float (FILEHANDLE *f*, char \* *item*, double *d*)**

Write a tag and its value to a configuration file.

**Parameters:**

*f* file handle  
*item* pointer to tag string  
*d* double value

**See also:**

[oapiWriteItem\\_string](#) for more details

**15.60.1.12 OAPIFUNC void oapiWriteItem\_int (FILEHANDLE *f*, char \* *item*, int *i*)**

Write a tag and its value to a configuration file.

**Parameters:**

*f* file handle  
*item* pointer to tag string  
*i* integer value

**See also:**

[oapiWriteItem\\_string](#) for more details

**15.60.1.13 OAPIFUNC void oapiWriteItem\_string (FILEHANDLE *f*, char \* *item*, char \* *string*)**

Write a tag and its value to a configuration file.

**Parameters:**

*f* file handle  
*item* pointer to tag string  
*string* character-string value

**Note:**

Use these functions to write items (tags and values) to configuration files.

The format of the written items is recognised by the corresponding **oapiReadItem\_xxx** functions.  
For historic reasons, the format for scenario file entries is different. Use the **oapiWriteLine** function.

**See also:**

[oapiReadItem\\_string](#)

**15.60.1.14 OAPIFUNC void oapiWriteItem\_vec (FILEHANDLE *f*, char \* *item*, const VECTOR3 & *vec*)**

Write a tag and its value to a configuration file.

**Parameters:**

*f* file handle  
*item* pointer to tag string  
*vec* vector value

**Note:**

Vector values are represented by space-separated triplets of floating point values.

**See also:**

[oapiWriteItem\\_string](#) for more details

**15.60.1.15 OAPIFUNC void oapiWriteLine (FILEHANDLE *file*, char \* *line*)**

Writes a line to a file.

**Parameters:**

*file* file handle  
*line* line to be written (zero-terminated)

**15.60.1.16 OAPIFUNC void oapiWriteLog (char \* *line*)**

Writes a line to the Orbiter log file (orbiter.log) in the main orbiter directory.

**Parameters:**

*line* line to be written (zero-terminated)

**Note:**

This function is intended for diagnostic initialisation and error messages by plugin modules. The messages should make it easier to track problems.

Avoid unnecessary output. In particular, don't write to the log file continuously from within the simulation loop.

**See also:**

[oapiWriteLogV](#)

**15.60.1.17 OAPIFUNC void oapiWriteLogV (const char \**format*, ...)**

Writes a formatted string with variable number of arguments to orbiter.log.

**Parameters:**

*format* Format string. Can contain any C-style parameter flags.

... List of output parameters. Must match the parameter flags in the format string.

**Note:**

A newline character is appended to the end of the format string.

**See also:**

[oapiWriteLog](#)

#### 15.60.1.18 OAPIFUNC void oapiWriteScenario\_float (FILEHANDLE *scn*, char \* *item*, double *d*)

Writes a floating point-valued item to a scenario file.

**Parameters:**

*scn* file handle

*item* item id

*d* floating point value to be written

#### 15.60.1.19 OAPIFUNC void oapiWriteScenario\_int (FILEHANDLE *scn*, char \* *item*, int *i*)

Writes an integer-valued item to a scenario file.

**Parameters:**

*scn* file handle

*item* item id

*i* integer value to be written

#### 15.60.1.20 OAPIFUNC void oapiWriteScenario\_string (FILEHANDLE *scn*, char \* *item*, char \* *string*)

Writes a string-valued item to a scenario file.

**Parameters:**

*scn* file handle

*item* item id

*string* string to be written (zero-terminated)

#### 15.60.1.21 OAPIFUNC void oapiWriteScenario\_vec (FILEHANDLE *scn*, char \* *item*, const VEC-TOR3 & *vec*)

Writes a vector-valued item to a scenario file.

**Parameters:**

*scn* file handle

*item* item id

*vec* vector to be written

## 15.61 Utility functions

### Functions

- OAPIFUNC double [oapiRand \(\)](#)  
*Returns uniformly distributed pseudo-random number in the range [0..1].*
- OAPIFUNC DWORD [oapiGetColour \(DWORD red, DWORD green, DWORD blue\)](#)  
*Returns a colour value adapted to the current screen colour depth for given red, green and blue components.*

#### 15.61.1 Function Documentation

##### 15.61.1.1 OAPIFUNC DWORD oapiGetColour (DWORD red, DWORD green, DWORD blue)

Returns a colour value adapted to the current screen colour depth for given red, green and blue components.

#### Parameters:

*red* red component (0-255)  
*green* green component (0-255)  
*blue* blue component (0-255)

#### Returns:

colour value

#### Note:

Colour values are required for some surface functions like `oapiClearSurface` or `oapiSetSurfaceColourKey`. The colour key for a given RGB triplet depends on the screen colour depth. This function returns the colour value for the closest colour match which can be displayed in the current screen mode.

In 24 and 32 bit modes the requested colour can always be matched. The colour value in that case is  $(\text{red} \ll 16) + (\text{green} \ll 8) + \text{blue}$ .

For 16 bit displays the colour value is calculated as  $((\text{red}*31)/255) \ll 11 + ((\text{green}*63)/255 \ll 5 + (\text{blue}*31)/255)$  assuming a "565" colour mode (5 bits for red, 6 for green, 5 for blue). This means that a requested colour may not be perfectly matched.

These colour values should not be used for Windows (GDI) drawing functions where a COLORREF value is expected.

##### 15.61.1.2 OAPIFUNC double oapiRand ()

Returns uniformly distributed pseudo-random number in the range [0..1].

#### Returns:

Random value between 0 and 1.

#### Note:

This function uses the system call `rand()`, so the quality of the random sequence depends on the system implementation. If you need high-quality random sequences you may need to implement your own generator.

Orbiter seeds the generator with the system time on startup, so the generated sequences are not reproducible.

## 15.62 User input functions

### Functions

- OAPIFUNC void **oapiOpenInputBox** (char \*title, bool(\*Clbk)(void \*, char \*, void \*), char \*buf=0, int vislen=20, void \*usrdata=0)  
*Opens a modal input box requesting a string from the user.*
- OAPIFUNC void **oapiOpenInputBoxEx** (const char \*title, bool(\*Clbk\_enter)(void \*, char \*, void \*), bool(\*Clbk\_cancel)(void \*, char \*, void \*), char \*buf=0, int vislen=20, void \*usrdata=0, DWORD flags=0)

### 15.62.1 Function Documentation

#### 15.62.1.1 OAPIFUNC void oapiOpenInputBox (char \* title, bool(\*)(void \*, char \*, void \*) Clbk, char \* buf = 0, int vislen = 20, void \* usrdta = 0)

Opens a modal input box requesting a string from the user.

##### Parameters:

*title* input box title  
*Clbk* callback function receiving the result of the user input (see notes)  
*buf* initial state of the input string  
*vislen* number of characters visible in input box  
*usrdta* user-defined data passed to the callback function

##### Note:

Format for callback function:

```
bool InputCallback (void *id, char *str, void *usrdata )
```

where id identifies the input box, str contains the user-supplied string, and usrdta contains the data specified in the call to oapiOpenInputBox. The callback function should return *true* if it accepts the string, false otherwise (the box will not be closed if the callback function returns false).

The box can be closed by the user by pressing Enter ("OK") or Esc ("Cancel"). The callback function is only called in the first case.

The input box is modal, i.e. all keyboard input is redirected into the dialog box. Normal key functions resume after the box is closed.

##### See also:

[oapiOpenInputBoxEx](#)

## 15.63 Onscreen annotations

### 15.63.1 Detailed Description

These functions can be used to display text on top of the render window during a running simulation. These may include flight parameters of the currently observed spacecraft, user instructions for tutorials, or debugging information during development.

## Functions

- OAPIFUNC NOTEHANDLE [oapiCreateAnnotation](#) (bool exclusive, double size, const VECTOR3 &col)  
*Creates an annotation handle for displaying onscreen text during a simulation.*
- OAPIFUNC bool [oapiDelAnnotation](#) (NOTEHANDLE hNote)  
*Deletes an annotation handle.*
- OAPIFUNC void [oapiAnnotationSetPos](#) (NOTEHANDLE hNote, double x1, double y1, double x2, double y2)  
*Resets the bounding box of the annotation display area.*
- OAPIFUNC void [oapiAnnotationSetSize](#) (NOTEHANDLE hNote, double size)  
*Resets the font size of the annotation text.*
- OAPIFUNC void [oapiAnnotationSetColour](#) (NOTEHANDLE hNote, const VECTOR3 &col)  
*Resets the font colour of the annotation text.*
- OAPIFUNC void [oapiAnnotationSetText](#) (NOTEHANDLE hNote, char \*note)  
*Writes a new annotation to screen, or overwrites the previous text.*

### 15.63.2 Function Documentation

#### 15.63.2.1 OAPIFUNC void [oapiAnnotationSetColour](#) (NOTEHANDLE *hNote*, const VECTOR3 & *col*)

Resets the font colour of the annotation text.

##### Parameters:

*hNote* annotation handle  
*col* font colour (RGB triplet with ranges 0-1)

##### See also:

[oapiCreateAnnotation](#)

#### 15.63.2.2 OAPIFUNC void [oapiAnnotationSetPos](#) (NOTEHANDLE *hNote*, double *x1*, double *y1*, double *x2*, double *y2*)

Resets the bounding box of the annotation display area.

##### Parameters:

*hNote* annotation handle  
*x1* left edge of bounding box ( $0 \leq x1 < x2$ )  
*y1* top edge of bounding box ( $0 \leq y1 < y2$ )  
*x2* right edge of bounding box ( $x1 < x2 \leq 1$ )  
*y2* bottom edge of bounding box ( $y1 < y2 \leq 1$ )

**Note:**

boundary values are specified in units of the render window area, with (0,0) being the top left corner, and (1,1) the bottom right corner.

If the bounding box is set too small, part of the annotation may not be visible.

**See also:**

[oapiCreateAnnotation](#)

**15.63.2.3 OAPIFUNC void oapiAnnotationSetSize (NOTEHANDLE *hNote*, double *size*)**

Resets the font size of the annotation text.

**Parameters:**

*hNote* annotation handle

*size* font size in relative units (> 0)

**Note:**

Annotations are sized in relation to the simulation window size. Size 1 is the default annotation size.

**See also:**

[oapiCreateAnnotation](#)

**15.63.2.4 OAPIFUNC void oapiAnnotationSetText (NOTEHANDLE *hNote*, char \* *note*)**

Writes a new annotation to screen, or overwrites the previous text.

**Parameters:**

*hNote* annotation handle

*note* annotation text

**See also:**

[oapiCreateAnnotation](#)

**15.63.2.5 OAPIFUNC NOTEHANDLE oapiCreateAnnotation (bool *exclusive*, double *size*, const VECTOR3 & *col*)**

Creates an annotation handle for displaying onscreen text during a simulation.

**Parameters:**

*exclusive* exclusive mode flag

*size* text scaling factor (>0, 1=standard)

*col* text colour (RGB triplet, range 0-1 for each component)

**Returns:**

Annotation handle

**See also:**

[oapiDelAnnotation](#), [oapiAnnotationSetPos](#), [oapiAnnotationSetSize](#), [oapiAnnotationSetColour](#), [oapiAnnotationSetText](#)

### 15.63.2.6 OAPIFUNC bool oapiDelAnnotation (NOTEHANDLE hNote)

Deletes an annotation handle.

**Parameters:**

*hNote* annotation handle

**Returns:**

*true* on success, *false* if an annotation corresponding to hNote was not found.

**See also:**

[oapiCreateAnnotation](#)

## 15.64 Obsolete functions

### Functions

- OAPIFUNC OBJHANDLE [oapiGetStationByName](#) (char \*name)
  - OAPIFUNC OBJHANDLE [oapiGetStationByIndex](#) (int index)
  - OAPIFUNC DWORD [oapiGetStationCount](#) ()
  - OAPIFUNC BOOL [oapiGetAirspeedVector](#) (OBJHANDLE hVessel, VECTOR3 \*speedvec)
 

*Returns a vessel's airspeed vector w.r.t. the closest planet or moon in the local horizon's frame of reference.*
  - OAPIFUNC BOOL [oapiGetShipAirspeedVector](#) (OBJHANDLE hVessel, VECTOR3 \*speedvec)
 

*Returns a vessel's airspeed vector w.r.t. the closest planet or moon in the vessel's local frame of reference.*
  - OAPIFUNC BOOL [oapiGetFocusAirspeed](#) (double \*airspeed)
 

*Returns the current focus vessel's airspeed w.r.t. the closest planet or moon.*
  - OAPIFUNC BOOL [oapiGetFocusAirspeedVector](#) (VECTOR3 \*speedvec)
 

*Returns the current focus vessel's airspeed vector w.r.t. the closest planet or moon in the local horizon's frame of reference.*
  - OAPIFUNC BOOL [oapiGetFocusShipAirspeedVector](#) (VECTOR3 \*speedvec)
 

*Returns the current focus vessel's airspeed vector w.r.t. closest planet or moon in the vessel's local frame of reference.*
  - OAPIFUNC void [oapiGetAtmPressureDensity](#) (OBJHANDLE hVessel, double \*pressure, double \*density)
 

*Returns the atmospheric pressure and density caused by a planetary atmosphere at the current vessel position.*
  - OAPIFUNC void [oapiGetFocusAtmPressureDensity](#) (double \*pressure, double \*density)
 

*Returns the atmospheric pressure and density caused by a planetary atmosphere at the current focus vessel's position.*
  - OAPIFUNC bool [oapiAcceptDelayedKey](#) (char key, double interval)
  - OAPIFUNC int [oapiRegisterMFDMode](#) (MFDMODESPEC &spec)
 

*Register a custom MFD mode.*
  - OAPIFUNC int [oapiGetMFDModeSpec](#) (char \*name, MFDMODESPEC \*\*spec=0)
 

*Returns the mode identifier and spec for an MFD mode defined by its name.*

### 15.64.1 Function Documentation

#### 15.64.1.1 OAPIFUNC BOOL oapiGetAirspeedVector (OBJHANDLE *hVessel*, VECTOR3 \* *speedvec*)

Returns a vessel's airspeed vector w.r.t. the closest planet or moon in the local horizon's frame of reference.

##### Deprecated

This method has been replaced by [oapiGetAirspeedVector\(OBJHANDLE,REFFRAME,VECTOR3\\*\)](#)

#### 15.64.1.2 OAPIFUNC void oapiGetAtmPressureDensity (OBJHANDLE *hVessel*, double \* *pressure*, double \* *density*)

Returns the atmospheric pressure and density caused by a planetary atmosphere at the current vessel position.

##### Deprecated

This function has been replaced by [oapiGetAtm](#).

##### Parameters:

*hVessel* vessel handle

*pressure* pointer to variable receiving pressure value [Pa]

*density* pointer to variable receiving density value [kg/m<sup>3</sup> ]

##### Note:

Pressure and density are calculated using an exponential barometric equation, without accounting for local variations.

##### See also:

[oapiGetAtm](#)

#### 15.64.1.3 OAPIFUNC BOOL oapiGetFocusAirspeed (double \* *airspeed*)

Returns the current focus vessel's airspeed w.r.t. the closest planet or moon.

##### Deprecated

This method has been replaced by [oapiGetAirspeed\(\)](#)

#### 15.64.1.4 OAPIFUNC BOOL oapiGetFocusAirspeedVector (VECTOR3 \* *speedvec*)

Returns the current focus vessel's airspeed vector w.r.t. the closest planet or moon in the local horizon's frame of reference.

##### Deprecated

This method has been replaced by [oapiGetAirspeedVector\(OBJHANDLE,REFFRAME,VECTOR3\\*\)](#)

**15.64.1.5 OAPIFUNC void oapiGetFocusAtmPressureDensity (double \**pressure*, double \**density*)**

Returns the atmospheric pressure and density caused by a planetary atmosphere at the current focus vessel's position.

**Deprecated**

This function has been replaced by [oapiGetAtm](#).

**Parameters:**

*pressure* pointer to variable receiving pressure value [Pa]

*density* pointer to variable receiving density value [ $\text{kg}/\text{m}^3$  ]

**Note:**

Pressure and density are calculated using an exponential barometric equation, without accounting for local variations.

**See also:**

[oapiGetAtm](#)

**15.64.1.6 OAPIFUNC BOOL oapiGetFocusShipAirspeedVector (VECTOR3 \**speedvec*)**

Returns the current focus vessel's airspeed vector w.r.t. closest planet or moon in the vessel's local frame of reference.

**Deprecated**

This method has been replaced by [oapiGetAirspeedVector\(OBJHANDLE,REFFRAME,VECTOR3\\*\)](#)

**15.64.1.7 OAPIFUNC int oapiGetMFDModeSpec (char \**name*, MFDMODESPEC \*\**spec* = 0)**

Returns the mode identifier and spec for an [MFD](#) mode defined by its name.

**Deprecated**

This function has been replaced by [oapiGetMFDModeSpecEx](#)

**See also:**

[oapiGetMFDModeSpecEx](#)

**15.64.1.8 OAPIFUNC BOOL oapiGetShipAirspeedVector (OBJHANDLE *hVessel*, VECTOR3 \**speedvec*)**

Returns a vessel's airspeed vector w.r.t. the closest planet or moon in the vessel's local frame of reference.

**Deprecated**

This method has been replaced by [oapiGetAirspeedVector\(OBJHANDLE,REFFRAME,VECTOR3\\*\)](#)

**15.64.1.9 OAPIFUNC OBJHANDLE oapiGetStationByIndex (int *index*)****Deprecated**

Stations are no longer distinguished from vessels. This function does not perform any action other than writing a warning to the log file. Use [oapiGetVesselByIndex](#) instead.

**15.64.1.10 OAPIFUNC OBJHANDLE oapiGetStationByName (char \* *name*)****Deprecated**

Stations are no longer distinguished from vessels. This function does not perform any action other than writing a warning to the log file. Use [oapiGetVesselByName](#) instead.

**15.64.1.11 OAPIFUNC DWORD oapiGetStationCount ()****Deprecated**

Stations are no longer distinguished from vessels. This function always returns 0. Use [oapiGetVesselCount](#) instead.

**15.64.1.12 OAPIFUNC int oapiRegisterMFDMode (MFDMODESPEC & *spec*)**

Register a custom [MFD](#) mode.

**Deprecated**

This function has been replaced by [oapiRegisterMFDMode\(MFDMODESPECEX&\)](#).

**See also:**

[oapiRegisterMFDMode\(MFDMODESPECEX&\)](#)

**15.65 Keyboard key identifiers****Defines**

- #define **OAPI\_KEY\_ESCAPE** 0x01  
*Escape key.*
- #define **OAPI\_KEY\_1** 0x02  
*'1' key on main keyboard*
- #define **OAPI\_KEY\_2** 0x03  
*'2' key on main keyboard*
- #define **OAPI\_KEY\_3** 0x04  
*'3' key on main keyboard*
- #define **OAPI\_KEY\_4** 0x05

- #define OAPI\_KEY\_5 0x06  
*'4' key on main keyboard*
- #define OAPI\_KEY\_6 0x07  
*'5' key on main keyboard*
- #define OAPI\_KEY\_7 0x08  
*'6' key on main keyboard*
- #define OAPI\_KEY\_8 0x09  
*'7' key on main keyboard*
- #define OAPI\_KEY\_9 0x0A  
*'8' key on main keyboard*
- #define OAPI\_KEY\_0 0x0B  
*'9' key on main keyboard*
- #define OAPI\_KEY\_MINUS 0x0C  
*'-' key on main keyboard*
- #define OAPI\_KEY\_EQUALS 0x0D  
*'=' key on main keyboard*
- #define OAPI\_KEY\_BACK 0x0E  
*backspace key*
- #define OAPI\_KEY\_TAB 0x0F  
*tab key*
- #define OAPI\_KEY\_Q 0x10  
*'Q' key*
- #define OAPI\_KEY\_W 0x11  
*'W' key*
- #define OAPI\_KEY\_E 0x12  
*'E' key*
- #define OAPI\_KEY\_R 0x13  
*'R' key*
- #define OAPI\_KEY\_T 0x14  
*'T' key*
- #define OAPI\_KEY\_Y 0x15  
*'Y' key*

- #define **OAPI\_KEY\_U** 0x16  
*'U' key*
- #define **OAPI\_KEY\_I** 0x17  
*'I' key*
- #define **OAPI\_KEY\_O** 0x18  
*'O' key*
- #define **OAPI\_KEY\_P** 0x19  
*'P' key*
- #define **OAPI\_KEY\_LBRACKET** 0x1A  
*'[' (left bracket) key*
- #define **OAPI\_KEY\_RBRACKET** 0x1B  
*']' (right bracket) key*
- #define **OAPI\_KEY\_RETURN** 0x1C  
*'Enter' key on main keyboard*
- #define **OAPI\_KEY\_LCONTROL** 0x1D  
*Left 'Ctrl' key.*
- #define **OAPI\_KEY\_A** 0x1E  
*'A' key*
- #define **OAPI\_KEY\_S** 0x1F  
*'S' key*
- #define **OAPI\_KEY\_D** 0x20  
*'D' key*
- #define **OAPI\_KEY\_F** 0x21  
*'F' key*
- #define **OAPI\_KEY\_G** 0x22  
*'G' key*
- #define **OAPI\_KEY\_H** 0x23  
*'H' key*
- #define **OAPI\_KEY\_J** 0x24  
*'J' key*
- #define **OAPI\_KEY\_K** 0x25  
*'K' key*
- #define **OAPI\_KEY\_L** 0x26  
*'L' key*

- #define **OAPI\_KEY\_SEMICOLON** 0x27  
';' (*semicolon*) key
- #define **OAPI\_KEY\_APOSTROPHE** 0x28  
' (*apostrophe*) key
- #define **OAPI\_KEY\_GRAVE** 0x29  
*accent grave*
- #define **OAPI\_KEY\_LSHIFT** 0x2A  
*Left 'Shift' key.*
- #define **OAPI\_KEY\_BACKSLASH** 0x2B  
\ (*Backslash*) key
- #define **OAPI\_KEY\_Z** 0x2C  
'Z' key
- #define **OAPI\_KEY\_X** 0x2D  
'X' key
- #define **OAPI\_KEY\_C** 0x2E  
'C' key
- #define **OAPI\_KEY\_V** 0x2F  
'V' key
- #define **OAPI\_KEY\_B** 0x30  
'B' key
- #define **OAPI\_KEY\_N** 0x31  
'N' key
- #define **OAPI\_KEY\_M** 0x32  
'M' key
- #define **OAPI\_KEY\_COMMA** 0x33  
, (*comma*) key
- #define **OAPI\_KEY\_PERIOD** 0x34  
. key on *main keyboard*
- #define **OAPI\_KEY\_SLASH** 0x35  
/' key on *main keyboard*
- #define **OAPI\_KEY\_RSHIFT** 0x36  
*Right 'Shift' key.*
- #define **OAPI\_KEY\_MULTIPLY** 0x37

\* on numeric keypad

- #define **OAPI\_KEY\_LALT** 0x38  
*left Alt*
- #define **OAPI\_KEY\_SPACE** 0x39  
*'Space' key*
- #define **OAPI\_KEY\_CAPITAL** 0x3A  
*caps lock key*
- #define **OAPI\_KEY\_F1** 0x3B  
*F1 function key.*
- #define **OAPI\_KEY\_F2** 0x3C  
*F2 function key.*
- #define **OAPI\_KEY\_F3** 0x3D  
*F3 function key.*
- #define **OAPI\_KEY\_F4** 0x3E  
*F4 function key.*
- #define **OAPI\_KEY\_F5** 0x3F  
*F5 function key.*
- #define **OAPI\_KEY\_F6** 0x40  
*F6 function key.*
- #define **OAPI\_KEY\_F7** 0x41  
*F7 function key.*
- #define **OAPI\_KEY\_F8** 0x42  
*F8 function key.*
- #define **OAPI\_KEY\_F9** 0x43  
*F9 function key.*
- #define **OAPI\_KEY\_F10** 0x44  
*F10 function key.*
- #define **OAPI\_KEY\_NUMLOCK** 0x45  
*'Num Lock' key*
- #define **OAPI\_KEY\_SCROLL** 0x46  
*Scroll lock.*
- #define **OAPI\_KEY\_NUMPAD7** 0x47  
*'7' key on numeric keypad*

- #define **OAPI\_KEY\_NUMPAD8** 0x48  
*'8' key on numeric keypad*
- #define **OAPI\_KEY\_NUMPAD9** 0x49  
*'9' key on numeric keypad*
- #define **OAPI\_KEY\_SUBTRACT** 0x4A  
*'-' key on numeric keypad*
- #define **OAPI\_KEY\_NUMPAD4** 0x4B  
*'4' key on numeric keypad*
- #define **OAPI\_KEY\_NUMPAD5** 0x4C  
*'5' key on numeric keypad*
- #define **OAPI\_KEY\_NUMPAD6** 0x4D  
*'6' key on numeric keypad*
- #define **OAPI\_KEY\_ADD** 0x4E  
*'+' key on numeric keypad*
- #define **OAPI\_KEY\_NUMPAD1** 0x4F  
*'1' key on numeric keypad*
- #define **OAPI\_KEY\_NUMPAD2** 0x50  
*'2' key on numeric keypad*
- #define **OAPI\_KEY\_NUMPAD3** 0x51  
*'3' key on numeric keypad*
- #define **OAPI\_KEY\_NUMPAD0** 0x52  
*'0' key on numeric keypad*
- #define **OAPI\_KEY\_DECIMAL** 0x53  
*'.' key on numeric keypad*
- #define **OAPI\_KEY\_OEM\_102** 0x56  
*| < > on UK/German keyboards*
- #define **OAPI\_KEY\_F11** 0x57  
*F11 function key.*
- #define **OAPI\_KEY\_F12** 0x58  
*F12 function key.*
- #define **OAPI\_KEY\_NUMPADEENTER** 0x9C  
*Enter on numeric keypad.*
- #define **OAPI\_KEY\_RCONTROL** 0x9D  
*right Control key*

- #define **OAPI\_KEY\_DIVIDE** 0xB5  
*'/' key on numeric keypad*
- #define **OAPI\_KEY\_RALT** 0xB8  
*right Alt*
- #define **OAPI\_KEY\_HOME** 0xC7  
*Home on cursor keypad.*
- #define **OAPI\_KEY\_UP** 0xC8  
*up-arrow on cursor keypad*
- #define **OAPI\_KEY\_PRIOR** 0xC9  
*PgUp on cursor keypad.*
- #define **OAPI\_KEY\_LEFT** 0xCB  
*left-arrow on cursor keypad*
- #define **OAPI\_KEY\_RIGHT** 0xCD  
*right-arrow on cursor keypad*
- #define **OAPI\_KEY\_END** 0xCF  
*End on cursor keypad.*
- #define **OAPI\_KEY\_DOWN** 0xD0  
*down-arrow on cursor keypad*
- #define **OAPI\_KEY\_NEXT** 0xD1  
*PgDn on cursor keypad.*
- #define **OAPI\_KEY\_INSERT** 0xD2  
*Insert on cursor keypad.*
- #define **OAPI\_KEY\_DELETE** 0xD3  
*Delete on cursor keypad.*

## 15.66 Logical key ids

### Defines

- #define **OAPI\_LKEY\_CockpitRotateLeft** 0  
*rotate camera left in cockpit view*
- #define **OAPI\_LKEY\_CockpitRotateRight** 1  
*rotate camera right in cockpit view*
- #define **OAPI\_LKEY\_CockpitRotateUp** 2

- #define OAPI\_LKEY\_CockpitRotateUp 1  
*rotate camera up in cockpit view*
- #define OAPI\_LKEY\_CockpitRotateDown 3  
*rotate camera down in cockpit view*
- #define OAPI\_LKEY\_CockpitDontLean 4  
*return to default cockpit camera position*
- #define OAPI\_LKEY\_CockpitLeanForward 5  
*move cockpit camera forward*
- #define OAPI\_LKEY\_CockpitLeanLeft 6  
*move cockpit camera left*
- #define OAPI\_LKEY\_CockpitLeanRight 7  
*move cockpit camera right*
- #define OAPI\_LKEY\_CockpitResetCam 8  
*rotate and shift cockpit camera back to default*
- #define OAPI\_LKEY\_PanelShiftLeft 9  
*shift 2D instrument panel left*
- #define OAPI\_LKEY\_PanelShiftRight 10  
*shift 2D instrument panel right*
- #define OAPI\_LKEY\_PanelShiftUp 11  
*shift 2D instrument panel up*
- #define OAPI\_LKEY\_PanelShiftDown 12  
*shift 2D instrument panel down*
- #define OAPI\_LKEY\_PanelSwitchLeft 13  
*switch to left neighbour panel*
- #define OAPI\_LKEY\_PanelSwitchRight 14  
*switch to right neighbour panel*
- #define OAPI\_LKEY\_PanelSwitchUp 15  
*switch to upper neighbour panel*
- #define OAPI\_LKEY\_PanelSwitchDown 16  
*switch to lower neighbour panel*
- #define OAPI\_LKEY\_TrackRotateLeft 17  
*turn track view camera left*
- #define OAPI\_LKEY\_TrackRotateRight 18  
*turn track view camera right*

- #define **OAPI\_LKEY\_TrackRotateUp** 19  
*turn track view camera up*
- #define **OAPI\_LKEY\_TrackRotateDown** 20  
*turn track view camera down*
- #define **OAPI\_LKEY\_TrackAdvance** 21  
*advance track view camera towards target*
- #define **OAPI\_LKEY\_TrackRetreat** 22  
*retreat track view camera from target*
- #define **OAPI\_LKEY\_GroundTiltLeft** 23  
*tilt camera left in ground view*
- #define **OAPI\_LKEY\_GroundTiltRight** 24  
*tilt camera right in ground view*
- #define **OAPI\_LKEY\_GroundTiltUp** 25  
*tilt camera up in ground view*
- #define **OAPI\_LKEY\_GroundTiltDown** 26  
*tilt camera down in ground view*
- #define **OAPI\_LKEY\_IncMainThrust** 27  
*increment thrust of main thrusters*
- #define **OAPI\_LKEY\_DecMainThrust** 28  
*decrement thrust of main thrusters*
- #define **OAPI\_LKEY\_KillMainRetro** 29  
*kill main and retro thrusters*
- #define **OAPI\_LKEY\_FullMainThrust** 30  
*temporary full main thrust*
- #define **OAPI\_LKEY\_FullRetroThrust** 31  
*temporary full retro thrust*
- #define **OAPI\_LKEY\_IncHoverThrust** 32  
*increment thrust of hover thrusters*
- #define **OAPI\_LKEY\_DecHoverThrust** 33  
*decrement thrust of hover thrusters*
- #define **OAPI\_LKEY\_RCSEnable** 34  
*enable/disable RCS (reaction control system)*
- #define **OAPI\_LKEY\_RCSMode** 35  
*toggle linear/rotational RCS mode*

- #define **OAPI\_LKEY\_RCSPitchUp** 36  
*rotational RCS: pitch up*
- #define **OAPI\_LKEY\_RCSPitchDown** 37  
*rotational RCS: pitch down*
- #define **OAPI\_LKEY\_RCSYawLeft** 38  
*rotational RCS: yaw left*
- #define **OAPI\_LKEY\_RCSYawRight** 39  
*rotational RCS: yaw right*
- #define **OAPI\_LKEY\_RCSBankLeft** 40  
*rotational RCS: bank left*
- #define **OAPI\_LKEY\_RCSBankRight** 41  
*rotational RCS: bank right*
- #define **OAPI\_LKEY\_RCSUp** 42  
*linear RCS: accelerate up (+y)*
- #define **OAPI\_LKEY\_RCSDown** 43  
*linear RCS: accelerate down (-y)*
- #define **OAPI\_LKEY\_RCSLeft** 44  
*linear RCS: accelerate left (-x)*
- #define **OAPI\_LKEY\_RCSRigh**t 45  
*linear RCS: accelerate right (+x)*
- #define **OAPI\_LKEY\_RCSForward** 46  
*linear RCS: accelerate forward (+z)*
- #define **OAPI\_LKEY\_RCSBack** 47  
*linear RCS: accelerate backward (-z)*
- #define **OAPI\_LKEY\_LPRCSPitchUp** 48  
*rotational RCS: pitch up 10%*
- #define **OAPI\_LKEY\_LPRCSPitchDown** 49  
*rotational RCS: pitch down 10%*
- #define **OAPI\_LKEY\_LPRCSYawLeft** 50  
*rotational RCS: yaw left 10%*
- #define **OAPI\_LKEY\_LPRCSYawRight** 51  
*rotational RCS: yaw right 10%*
- #define **OAPI\_LKEY\_LPRCSBankLeft** 52

*rotational RCS: bank left 10%*

- #define **OAPI\_LKEY\_LPRCSBankRight** 53

*rotational RCS: bank right 10%*

- #define **OAPI\_LKEY\_LPRCSUp** 54

*linear RCS: accelerate up 10% (+y)*

- #define **OAPI\_LKEY\_LPRCSDown** 55

*linear RCS: accelerate down 10% (-y)*

- #define **OAPI\_LKEY\_LPRCSLeft** 56

*linear RCS: accelerate left 10% (-x)*

- #define **OAPI\_LKEY\_LPRCSRRight** 57

*linear RCS: accelerate right 10% (+x)*

- #define **OAPI\_LKEY\_LPRCSForward** 58

*linear RCS: accelerate forward 10% (+z)*

- #define **OAPI\_LKEY\_LPRCSBack** 59

*linear RCS: accelerate backward 10% (-z)*

- #define **OAPI\_LKEY\_NMHoldAltitude** 60

*toggle navmode: hold altitude*

- #define **OAPI\_LKEY\_NMHLevel** 61

*toggle navmode: level with horizon*

- #define **OAPI\_LKEY\_NMPrograde** 62

*toggle navmode: prograde*

- #define **OAPI\_LKEY\_NMRetrograde** 63

*toggle navmode: retrograde*

- #define **OAPI\_LKEY\_NMNormal** 64

*toggle navmode: normal to orbital plane*

- #define **OAPI\_LKEY\_NMAntinormal** 65

*toggle navmode: antinormal to orbital plane*

- #define **OAPI\_LKEY\_NMKillrot** 66

*toggle navmode: kill rotation*

- #define **OAPI\_LKEY\_Undock** 67

*undock from docked vessel*

- #define **OAPI\_LKEY\_IncElevatorTrim** 68

*increment elevator trim setting*

- #define **OAPI\_LKEY\_DecElevatorTrim** 69  
*decrement elevator trim setting*
- #define **OAPI\_LKEY\_WheelbrakeLeft** 70  
*apply wheelbrake left*
- #define **OAPI\_LKEY\_WheelbrakeRight** 71  
*apply wheelbrake right*
- #define **OAPI\_LKEY\_HUD** 72  
*toggle HUD on/off*
- #define **OAPI\_LKEY\_HUDMode** 73  
*switch through HUD modes*
- #define **OAPI\_LKEY\_HUDReference** 74  
*query reference object for HUD display*
- #define **OAPI\_LKEY\_HUDTarget** 75  
*query target object for HUD display*
- #define **OAPI\_LKEY\_HUDColour** 76  
*switch through HUD colours*
- #define **OAPI\_LKEY\_IncSimSpeed** 77  
*increase simulation speed x10*
- #define **OAPI\_LKEY\_DecSimSpeed** 78  
*decrease simulation speed x0.1*
- #define **OAPI\_LKEY\_IncFOV** 79  
*increment field of view*
- #define **OAPI\_LKEY\_DecFOV** 80  
*decrement field of view*
- #define **OAPI\_LKEY\_StepIncFOV** 81  
*increment field of view by 10 deg*
- #define **OAPI\_LKEY\_StepDecFOV** 82  
*decrement field of view by 10 deg*
- #define **OAPI\_LKEY\_MainMenu** 83  
*open main menu*
- #define **OAPI\_LKEY\_DlgHelp** 84  
*open help dialog*
- #define **OAPI\_LKEY\_DlgCamera** 85  
*open camera dialog*

- #define **OAPI\_LKEY\_DlgSimspeed** 86  
*open simulation speed dialog*
- #define **OAPI\_LKEY\_DlgCustomCmd** 87  
*open custom command dialog*
- #define **OAPI\_LKEY\_DlgVisHelper** 88  
*open visual helper dialog*
- #define **OAPI\_LKEY\_DlgRecorder** 89  
*open flight recorder dialog*
- #define **OAPI\_LKEY\_DlgInfo** 90  
*open object info dialog*
- #define **OAPI\_LKEY\_DlgMap** 91  
*open map dialog*
- #define **OAPI\_LKEY\_DlgNavaid** 92  
*open nav transmitter list*
- #define **OAPI\_LKEY\_ToggleInfo** 93  
*OBSOLETE.*
- #define **OAPI\_LKEY\_ToggleFPS** 94  
*OBSOLETE.*
- #define **OAPI\_LKEY\_ToggleCamInternal** 95  
*switch between cockpit and external camera*
- #define **OAPI\_LKEY\_ToggleTrackMode** 96  
*switch between track camera modes*
- #define **OAPI\_LKEY\_TogglePanelMode** 97  
*switch between cockpit modes*
- #define **OAPI\_LKEY\_TogglePlanetarium** 98  
*toggle celestial marker display on/off*
- #define **OAPI\_LKEY\_ToggleRecPlay** 99  
*toggle flight recorder/playback on/off*
- #define **OAPI\_LKEY\_Pause** 100  
*toggle simulation pause on/off*
- #define **OAPI\_LKEY\_Quicksave** 101  
*quick-save current simulation state*
- #define **OAPI\_LKEY\_Quit** 102

*quit simulation session*

- #define [OAPI\\_LKEY\\_DlgSelectVessel](#) 103  
*open vessel selection dialog*
- #define [OAPI\\_LKEY\\_SelectPrevVessel](#) 104  
*switch focus to previous vessel*
- #define [LKEY\\_COUNT](#) 105  
*number of logical key definitions*

## 15.67 Top-level module callback functions

### 15.67.1 Detailed Description

This section contains a list of global nonmember callback functions that can be defined by an addon module. Orbiter will call these functions when specific events occur, e.g. a module is activated or deactivated, a simulation session is opened or closed, etc.

#### Modules

- [General module callback functions](#)
- [Vessel module callback functions](#)
- [Plugin module callback functions](#)

## 15.68 General module callback functions

### 15.68.1 Detailed Description

Module initialisation and exit notifications. The two callback functions in this group are called by Orbiter when the module is loaded or unloaded, respectively. It is used for all module types (plugin and vessel modules).

#### Functions

- DLLCLBK void [InitModule](#) (HINSTANCE hModule)  
*Module initialisation callback function.*
- DLLCLBK void [ExitModule](#) (HINSTANCE hModule)  
*Module exit notification callback function.*

### 15.68.2 Function Documentation

#### 15.68.2.1 DLLCLBK void [ExitModule](#) (HINSTANCE *hModule*)

Module exit notification callback function.

**Parameters:**

*hModule* module handle

**Note:**

This function is called by Orbiter when a module is deactivated.

For plugin modules, ExitModule is called at program shutdown for all active modules, or whenever a user deactivates a module in the *Modules* tab of the Orbiter launchpad.

For vessel modules, ExitModule is called when a simulation session is closed for any vessel types active at that time, or during a session when the last vessel of this type is destroyed.

**15.68.2.2 DLLCLBK void InitModule (HINSTANCE *hModule*)**

Module initialisation callback function.

**Parameters:**

*hModule* module handle

**Note:**

This function is called by Orbiter when a module becomes active.

For plugin modules, InitModule is called at program start for all modules in the *active module list* of orbiter.def, or whenever a user activates a module in the *Modules* tab of the Orbiter launchpad.

For vessel modules, InitModule is called whenever the first vessel of the corresponding type is created (usually at the start of a simulation session, or during a simulation if the first vessel instance is created dynamically).

*hModule* is the module handle that identifies the addon DLL being initialised. It can be stored and used later, e.g. for loading resources from the module. To get the handle of the Orbiter core module, use [oapiGetOrbiterInstance](#).

## 15.69 Vessel module callback functions

### 15.69.1 Detailed Description

This section contains a list of nonmember callback functions for vessel modules. Apart from the general module initialisation and exit functions in [Top-level module callback functions](#), the only vessel-specific top-level callback functions are notifications for vessel creation and deletion. During the vessel creation callback, the module should create an instance of a class derived from [VESSEL2](#) or [VESSEL3](#), and delete the instance during the vessel deletion callback. All other events should be handled by overloading the appropriate [VESSEL2](#) and [VESSEL3](#) member callback functions.

#### Functions

- DLLCLBK [VESSEL](#) \* ovcInit (OBJHANDLE hvessel, int flightmodel)  
*Vessel instance creation notification.*
- DLLCLBK void ovcExit ([VESSEL](#) \*vessel)  
*Vessel deletion notification.*

### 15.69.2 Function Documentation

#### 15.69.2.1 DLLCLBK void ovcExit (**VESSEL** \* *vessel*)

Vessel deletion notification.

##### Parameters:

*vessel* pointer to vessel instance

##### Note:

This function is called by Orbiter whenever a vessel of the type defined by the module is about to be destroyed at the end or during a simulation session.

The pointer passed to the function is the same as the one returned by ovcInit for the corresponding vessel.

Typically, the implementation of this function should cast the pointer to a pointer to the derived vessel class, and delete the object.

##### Examples:

[VESSEL2.cpp](#).

#### 15.69.2.2 DLLCLBK **VESSEL**\* ovcInit (OBJHANDLE *hvessel*, int *flightmodel*)

Vessel instance creation notification.

##### Parameters:

*hvessel* vessel handle

*flightmodel* flight model selection identifier

##### Returns:

The function should return a pointer to the derived **VESSEL** instance it created.

##### Note:

This function is called by Orbiter whenever a vessel of the type defined by the module is created at the beginning or during a simulation session.

The implementation should create an instance of a vessel class derived from **VESSEL**, **VESSEL2** or **VESSEL3** and return a pointer to it.

*hvessel* is a handle that identifies the vessel instance in Orbiter.

*flightmodel* identifies the realism level of the requested flight model. This value may be 0 (simple) or 1 (complex). Vessel implementation that support different flight models for easy/realistic setups can use this value to define the appropriate model.

##### Examples:

[VESSEL2.cpp](#).

## 15.70 Plugin module callback functions

### 15.70.1 Detailed Description

The callback functions in this group are specific for *plugin* modules, i.e. modules that can be activated or deactivated in the Modules tab of the Orbiter Launchpad. They can not be used in vessel modules.

Note that most of the top-level plugin callback functions (opcXXX) are now obsolete and should no longer be used. Addon modules should instead create an instance of a class derived from the [oapi::Module](#) class during [InitModule](#), and overload the appropriate class-level callback functions.

## Functions

- DLLCLBK void [opcOpenRenderViewport](#) (HWND hRenderWindow, DWORD width, DWORD height, BOOL fullscreen)  
*Called by Orbiter when a graphics-enabled simulation session is started.*
- DLLCLBK void [opcCloseRenderViewport](#) ()  
*Called by Orbiter when a graphics-enabled simulation session is closed.*
- DLLCLBK void [opcPreStep](#) (double simt, double simdt, double mjd)  
*Time step notification before state update.*
- DLLCLBK void [opcPostStep](#) (double simt, double simdt, double mjd)  
*Time step notification after state update.*
- DLLCLBK void [opcFocusChanged](#) (OBJHANDLE hGainsFocus, OBJHANDLE hLosesFocus)  
*Change of input focus notification.*
- DLLCLBK void [opcTimeAccChanged](#) (double new\_warp, double old\_warp)  
*Change of time acceleration notification.*
- DLLCLBK void [opcPause](#) (bool pause)  
*Simulation pause/resume notification.*
- DLLCLBK void [opcDeleteVessel](#) (OBJHANDLE hVessel)  
*Vessel destruction notification.*

### 15.70.2 Function Documentation

#### 15.70.2.1 DLLCLBK void [opcCloseRenderViewport](#) ()

Called by Orbiter when a graphics-enabled simulation session is closed.

##### Deprecated

This function has been replaced by [oapi::Module::clbkSimulationEnd](#).

##### Note:

Plugins should no longer implement this function. Instead they should create an instance of a class derived from [oapi::Module](#) during [InitModule](#) that overloads the [oapi::Module::clbkSimulationEnd](#) method, and register it with [oapiRegisterModule](#).

[opcCloseRenderViewport](#) is called by Orbiter only if no instance of [oapi::Module](#) is created and registered during [InitModule](#), or if a registered module does not overload the [oapi::Module::clbkSimulationEnd](#) method.

**15.70.2.2 DLLCLBK void opcDeleteVessel (OBJHANDLE *hVessel*)**

Vessel destruction notification.

Sent to modules immediately before a vessel is destroyed. After this callback method returns, the object handle (*hVessel*) and will no longer be valid. Modules should make sure that they don't access the vessel in any form after this point.

**Deprecated**

This function has been replaced by [oapi::Module::clbkDeleteVessel](#).

**Parameters:**

*hVessel* object handle for the vessel being destroyed.

**Note:**

Plugins should no longer implement this function. Instead they should create an instance of a class derived from [oapi::Module](#) during [InitModule](#) that overloads the [oapi::Module::clbkDeleteVessel](#) method, and register it with [oapiRegisterModule](#).

`opcDeleteVessel` is called by Orbiter only if no instance of [oapi::Module](#) is created and registered during [InitModule](#), or if a registered module does not overload the [oapi::Module::clbkDeleteVessel](#) method.

**15.70.2.3 DLLCLBK void opcFocusChanged (OBJHANDLE *hGainsFocus*, OBJHANDLE *hLosesFocus*)**

Change of input focus notification.

Called when input focus (keyboard and joystick control) is switched to a new vessel (for example as a result of a call to [oapiSetFocus](#)).

**Deprecated**

This function has been replaced by [oapi::Module::clbkFocusChanged](#).

**Parameters:**

*hGainsFocus* handle of vessel receiving the input focus

*hLosesFocus* handle of vessel losing focus

**Note:**

Plugins should no longer implement this function. Instead they should create an instance of a class derived from [oapi::Module](#) during [InitModule](#) that overloads the [oapi::Module::clbkFocusChanged](#) method, and register it with [oapiRegisterModule](#).

`opcFocusChanged` is called by Orbiter only if no instance of [oapi::Module](#) is created and registered during [InitModule](#), or if a registered module does not overload the [oapi::Module::clbkFocusChanged](#) method.

**15.70.2.4 DLLCLBK void opcOpenRenderViewport (HWND *hRenderWindow*, DWORD *width*, DWORD *height*, BOOL *fullscreen*)**

Called by Orbiter when a graphics-enabled simulation session is started.

**Deprecated**

This function has been replaced by [oapi::Module::clbkSimulationStart](#).

**Parameters:**

*hRenderWindow* render window handle  
*width* viewport width [pixel]  
*height* viewport height [pixel]  
*fullscreen* flag for fullscreen mode

**Note:**

Plugins should no longer implement this function. Instead they should create an instance of a class derived from [oapi::Module](#) during [InitModule](#) that overloads the [oapi::Module::clbkSimulationStart](#) method, and register it with [oapiRegisterModule](#).

[opcOpenRenderViewport](#) is called by Orbiter only if no instance of [oapi::Module](#) is created and registered during [InitModule](#), or if a registered module does not overload the [oapi::Module::clbkSimulationStart](#) method.

**15.70.2.5 DLLCLBK void opcPause (bool *pause*)**

Simulation pause/resume notification.

Called when the pause/resume state of the simulation has changed.

**Deprecated**

This function has been replaced by [oapi::Module::clbkPause](#).

**Parameters:**

*pause* pause/resume state: true if simulation has been paused, false if simulation has been resumed.

**Note:**

Plugins should no longer implement this function. Instead they should create an instance of a class derived from [oapi::Module](#) during [InitModule](#) that overloads the [oapi::Module::clbkPause](#) method, and register it with [oapiRegisterModule](#).

[opcPause](#) is called by Orbiter only if no instance of [oapi::Module](#) is created and registered during [InitModule](#), or if a registered module does not overload the [oapi::Module::clbkPause](#) method.

**15.70.2.6 DLLCLBK void opcPostStep (double *simt*, double *simdt*, double *mjd*)**

Time step notification after state update.

Called at each time step of the simulation, after the state has been updated to the current simulation time.

**Deprecated**

This function has been replaced by [oapi::Module::clbkPostStep](#).

**Parameters:**

*simt* current simulation time [s]

*simdt* length of the last time step [s]

*mjd* simulation time in Modified Julian Date format [days]

**Note:**

Plugins should no longer implement this function. Instead they should create an instance of a class derived from [oapi::Module](#) during [InitModule](#) that overloads the [oapi::Module::clbkPostStep](#) method, and register it with [oapiRegisterModule](#).

[opcPostStep](#) is called by Orbiter only if no instance of [oapi::Module](#) is created and registered during [InitModule](#), or if a registered module does not overload the [oapi::Module::clbkPostStep](#) method.

#### 15.70.2.7 DLLCLBK void [opcPreStep](#) (double *simt*, double *simdt*, double *mjd*)

Time step notification before state update.

Called at each time step of the simulation, before the state is updated to the current simulation time. This function is only called when the "physical" state of the simulation is propagated in time. [opcPreStep](#) is not called while the simulation is paused, even if the user moves the camera.

**Deprecated**

This function has been replaced by [oapi::Module::clbkPreStep](#).

**Parameters:**

*simt* simulation time after the currently processed step [s]

*simdt* length of the currently processed step [s]

*mjd* simulation time after the currently processed step in Modified Julian Date format [days]

**Note:**

Plugins should no longer implement this function. Instead they should create an instance of a class derived from [oapi::Module](#) during [InitModule](#) that overloads the [oapi::Module::clbkPreStep](#) method, and register it with [oapiRegisterModule](#).

[opcPreStep](#) is called by Orbiter only if no instance of [oapi::Module](#) is created and registered during [InitModule](#), or if a registered module does not overload the [oapi::Module::clbkPreStep](#) method.

#### 15.70.2.8 DLLCLBK void [opcTimeAccChanged](#) (double *new\_warp*, double *old\_warp*)

Change of time acceleration notification.

Called when the simulation time acceleration factor changes.

**Deprecated**

This function has been replaced by [oapi::Module::clbkTimeAccChanged](#).

**Parameters:**

*new\_warp* new time acceleration factor

*old\_warp* old time acceleration factor

**Note:**

Plugins should no longer implement this function. Instead they should create an instance of a class derived from [oapi::Module](#) during [InitModule](#) that overloads the [oapi::Module::clbkTimeAccChanged](#) method, and register it with [oapiRegisterModule](#).

[opcTimeAccChanged](#) is called by Orbiter only if no instance of [oapi::Module](#) is created and registered during [InitModule](#), or if a registered module does not overload the [oapi::Module::clbkTimeAccChanged](#) method.

## 16 Class Documentation

### 16.1 ANIMATION Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for ANIMATION:

#### 16.1.1 Detailed Description

Animation definition.

Defines a complete animation, including a list of components, the current animation state, and the default state (as represented by the original mesh).

#### Public Attributes

- double [defstate](#)  
*default animation state in the mesh*
- double [state](#)  
*current state*
- [UINT ncomp](#)  
*number of components*
- [ANIMATIONCOMP \\*\\* comp](#)  
*list of components*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 16.2 ANIMATIONCOMP Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for ANIMATIONCOMP:



```
graph TD; V[VESSEL] --> AC[ANIMATIONCOMP]; AC --> MT[MGROUP_TRANSFORM]; AC --> P[ANIMATIONCOMP]; AC --> C[ANIMATIONCOMP]; AC --> NC[UINT];
```

### 16.2.1 Detailed Description

Animation component definition.

Defines one component of an animation, including the mesh transformation, the relative start and end points within the entire animation, and any parent and child relationships with other animations.

**See also:**

[VESSEL::AddAnimationComponent](#)

#### Public Attributes

- double **state0**  
*first end state*
- double **state1**  
*second end state*
- MGROUP\_TRANSFORM \* **trans**  
*transformation*
- ANIMATIONCOMP \* **parent**  
*parent transformation*
- ANIMATIONCOMP \*\* **children**  
*list of children*
- UINT **nchildren**  
*number of children*

The documentation for this struct was generated from the following file:

- OrbiterSDK/include/OrbiterAPI.h

## 16.3 ATMCONST Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for ATMCONST:

### 16.3.1 Detailed Description

Planetary atmospheric constants structure.

#### Public Attributes

- double [p0](#)  
*pressure at mean radius ('sea level') [Pa]*
- double [rho0](#)  
*density at mean radius*
- double [R](#)  
*specific gas constant [J/(K kg)]*
- double [gamma](#)  
*ratio of specific heats,  $c_p/c_v$*
- double [C](#)  
*exponent for pressure equation (temporary)*
- double [O2pp](#)  
*partial pressure of oxygen*
- double [altspace](#)  
*atmosphere altitude limit [m]*
- double [radlimit](#)  
*radius limit (altspace + mean radius)*
- double [horizonalt](#)  
*horizon rendering altitude*
- [VECTOR3 color0](#)  
*sky colour at sea level during daytime*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 16.4 ATMOSPHERE Class Reference

```
#include <CelBodyAPI.h>
```

Collaboration diagram for ATMOSPHERE:

### 16.4.1 Detailed Description

Defines the physical atmospheric properties for a celestial body.

#### See also:

[CELBODY2](#)

#### Public Types

- enum [PRM\\_IN\\_FLAG](#) {  
    [PRM\\_ALT](#) = 0x0001, [PRM\\_LNG](#) = 0x0002, [PRM\\_LAT](#) = 0x0004, [PRM\\_FBR](#) = 0x0008,  
    [PRM\\_F](#) = 0x0010, [PRM\\_AP](#) = 0x0020 }  
    *Parameter flags for atmospheric data input.*

#### Public Member Functions

- [ATMOSPHERE \(CELBODY2 \\*body\)](#)  
*Constructor. Creates an atmosphere instance for 'body'.*
- virtual const char \* [clbkName \(\) const](#) =0  
*A brief name that identifies the atmosphere model.*
- virtual bool [clbkConstants \(ATMCONST \\*atmc\) const](#)  
*Returns some general properties of the atmosphere.*
- virtual bool [clbkParams \(const PRM\\_IN \\*prm\\_in, PRM\\_OUT \\*prm\\_out\)](#)  
*Called by Orbiter to obtain atmospheric parameters for a given set of input parameters at the current simulation time.*

#### Protected Attributes

- [CELBODY2 \\* cbody](#)  
*associated celestial body instance*

## Classes

- struct [PRM\\_IN](#)  
*Input parameters for atmospheric data calculation.*
- struct [PRM\\_OUT](#)  
*Output parameters for atmospheric data calculation.*

### 16.4.2 Member Enumeration Documentation

#### 16.4.2.1 enum ATMOSPHERE::PRM\_IN\_FLAG

Parameter flags for atmospheric data input.

See also:

[ATMPRM\\_IN](#)

Enumerator:

- PRM\_ALT** altitude valid (otherwise use alt=0)
- PRM\_LNG** longitude valid (otherwise use lng=0)
- PRM\_LAT** latitude valid (otherwise use lat=0)
- PRM\_FBR** average flux valid (otherwise use f107avg=140)
- PRM\_F** current flux valid (otherwise use f107=f107avg)
- PRM\_AP** geomagnetic index valid (otherwise use ap=3)

### 16.4.3 Constructor & Destructor Documentation

#### 16.4.3.1 ATMOSPHERE::ATMOSPHERE (CELBODY2 \* *body*)

Constructor. Creates an atmosphere instance for 'body'.

Parameters:

*body* pointer to celestial body

### 16.4.4 Member Function Documentation

#### 16.4.4.1 virtual const char\* ATMOSPHERE::clbkName () const [pure virtual]

A brief name that identifies the atmosphere model.

Returns:

Pointer to persistent string buffer that contains the model name.

Note:

The returned name should not be longer than approx. 10 characters.

**16.4.4.2 virtual bool ATMOSPHERE::clbkConstants (ATMCONST \* *atmc*) const [virtual]**

Returns some general properties of the atmosphere.

**Parameters:**

*atmc* pointer to structure to be filled by clbkConstants

**Returns:**

*true* if paramters were supplied, *false* otherwise.

**Default action:**

Sets the following structure entries to default values:

- *atmc->R* = 286.91
- *atmc->gamma* = 1.4 but leaves the other values unchanged. Returns *false*.

**Note:**

This function should be overloaded to provide appropriate basic physical atmospheric properties, such as sea level density and pressure, gas constant, cutoff altitude, as well as rendering colour and rendering altitude.

For complex atmospheric models, some of the parameters in the [ATMCONST](#) structure may not be constants (e.g. ground density and pressure. In that case, the return values should be reasonable mean values.

Some of these values may be overwritten by configuration file settings.

**16.4.4.3 virtual bool ATMOSPHERE::clbkParams (const PRM\_IN \* *prm\_in*, PRM\_OUT \* *prm\_out*) [virtual]**

Called by Orbiter to obtain atmospheric parameters for a given set of input parameters at the current simulation time.

**Parameters:**

*prm\_in* input parameters for atmospheric data calculation (see [PRM\\_IN](#))

*prm\_out* returned data (see [PRM\\_OUT](#))

**Returns:**

*true* if atmospheric data were calculated and returned, *false* if the planet has no atmosphere or if the specified position is outside the supported distance of the atmospheric model.

**Default action:**

None, returns *false*.

The documentation for this class was generated from the following file:

- OrbiterSDK/include/CelBodyAPI.h

**16.5 ATMOSPHERE::PRM\_IN Struct Reference**

```
#include <CelBodyAPI.h>
```

### 16.5.1 Detailed Description

Input parameters for atmospheric data calculation.

**See also:**

[clbkAtmParam](#)

#### Public Attributes

- double [alt](#)  
*altitude [m]*
- double [lng](#)  
*longitude [rad]*
- double [lat](#)  
*latitude [rad]*
- double [f107bar](#)  
*average F10.7 flux over recent period*
- double [f107](#)  
*current F10.7 flux*
- double [ap](#)  
*magnetic index*
- DWORD [flag](#)  
*parameter flags (see [PRM\\_IN\\_FLAG](#))*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[CelBodyAPI.h](#)

## 16.6 ATMOSPHERE::PRM\_OUT Struct Reference

```
#include <CelBodyAPI.h>
```

### 16.6.1 Detailed Description

Output parameters for atmospheric data calculation.

**See also:**

[clbkAtmParam](#)

### Public Attributes

- double **T**  
*temperature [K]*
- double **p**  
*pressure [Pa]*
- double **rho**  
*density [kg/m<sup>3</sup>]*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[CeIBodyAPI.h](#)

## 16.7 ATMPARAM Struct Reference

```
#include <OrbiterAPI.h>
```

### 16.7.1 Detailed Description

Atmospheric parameters structure.

### Public Attributes

- double **T**  
*temperature [K]*
- double **p**  
*pressure [Pa]*
- double **rho**  
*density [kg/m<sup>3</sup>]*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 16.8 BEACONLIGHTSPEC Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for BEACONLIGHTSPEC:

### 16.8.1 Detailed Description

vessel beacon light parameters

#### Public Attributes

- **DWORD shape**  
*beacon shape identifier (see [Light beacon shape parameters](#))*
- **VECTOR3 \* pos**  
*pointer to position in vessel coordinates*
- **VECTOR3 \* col**  
*pointer to beacon RGB colour*
- **double size**  
*beacon radius*
- **double falloff**  
*distance falloff parameter*
- **double period**  
*strobe period (0 for continuous)*
- **double duration**  
*strobe duration*
- **double tofs**  
*strobe time offset*
- **bool active**  
*beacon lit?*

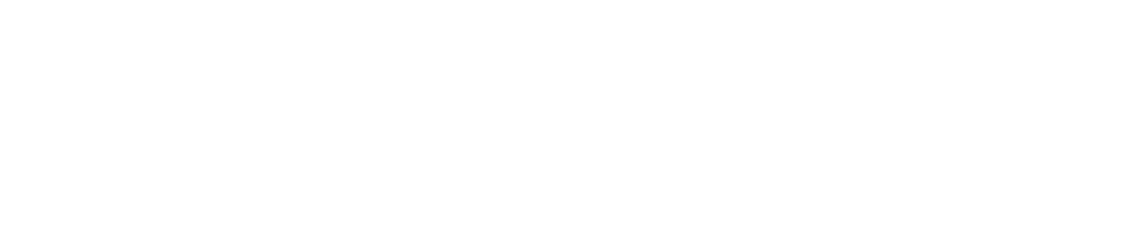
The documentation for this struct was generated from the following file:

- OrbiterSDK/include/[OrbiterAPI.h](#)

## 16.9 oapi::Brush Class Reference

```
#include <DrawAPI.h>
```

Inheritance diagram for oapi::Brush:



```
graph TD; Object --> Brush
```

Collaboration diagram for oapi::Brush:



```
graph TD; Brush --- Object
```

### 16.9.1 Detailed Description

A brush is a drawing resource for filling closed figures (rectangles, ellipses, polygons).

#### Public Member Functions

- virtual [~Brush \(\)](#)

*Brush destructor.*

#### Protected Member Functions

- [Brush \(DWORD col\)](#)

*Brush constructor.*

### 16.9.2 Constructor & Destructor Documentation

#### 16.9.2.1 oapi::Brush::Brush (DWORD col) [inline, protected]

[Brush](#) constructor.

##### Parameters:

*col* brush colour (format: 0xBBGGRR)

The documentation for this class was generated from the following file:

- Orbitersdk/include/[DrawAPI.h](#)

## 16.10 CELBODY Class Reference

```
#include <CelBodyAPI.h>
```

Inheritance diagram for CELBODY:

### 16.10.1 Detailed Description

This is the base class for celestial body classes.

**CELBODY** defines callback methods which Orbiter will call whenever it requires information from your planet module. You define the behaviour of the planet by overloading the relevant methods. Below is a list of public **CELBODY** methods:

#### See also:

[Planet Modules](#)

#### Public Member Functions

- int **Version** () const  
*Return version number.*
- virtual bool **bEphemeris** () const  
*Returns true or false depending on whether the module supports ephemeris calculation.*
- virtual void **clbkInit** (FILEHANDLE cfg)  
*Called when the planet is initialised at the beginning of a simulation run.*
- virtual int **clbkEphemeris** (double mjd, int req, double \*ret)  
*Called when Orbiter requires (non-sequential) ephemeris data from the planet for a given time.*
- virtual int **clbkFastEphemeris** (double simt, int req, double \*ret)  
*Called by Orbiter to update the body's state to the next simulation frame.*
- virtual bool **clbkAtmParam** (double alt, ATMPARAM \*prm)  
*Called by Orbiter to obtain atmospheric parameters at a given altitude.*

#### Protected Member Functions

- void **Pol2Crt** (double \*pol, double \*crt)  
*Convert from polar to cartesian coordinates.*

**Protected Attributes**

- short [version](#)  
*version number*

**16.10.2 Member Function Documentation****16.10.2.1 int CELBODY::Version () const [inline]**

Return version number.

**Returns:**

Version number (1 for [CELBODY](#), 2 for [CELBODY2](#))

**16.10.2.2 virtual bool CELBODY::bEphemeris () const [virtual]**

Returns *true* or *false* depending on whether the module supports ephemeris calculation.

**Returns:**

If your module supports ephemeris calculation (that is, if it defines the clbkEphemeris and clbkFastEphemeris methods) return *true*. Otherwise return *false*.

**Default action:**

Returns *false*.

**16.10.2.3 virtual void CELBODY::clbkInit (FILEHANDLE *cfg*) [virtual]**

Called when the planet is initialised at the beginning of a simulation run.

This function allows to read any parameters from the configuration file, and perform additional initialisation tasks such as reading data files.

**Parameters:**

*cfg* file handle of configuration file

**Default action:**

None.

Reimplemented in [CELBODY2](#).

**16.10.2.4 virtual int CELBODY::clbkEphemeris (double *mjd*, int *req*, double \* *ret*) [virtual]**

Called when Orbiter requires (non-sequential) ephemeris data from the planet for a given time.

**Parameters:**

*mjd* ephemeris date (days, in Modified Julian Date format)

*req* data request bitflags (see notes)

*ret* pointer to result vector

**Returns:**

bitflags describing returned data (see notes)

**Default action:**

None, returning 0

**Note:**

The ephemeris data should be calculated with respect to the body's parent body, in the ecliptic frame (J2000 equator and equinox).

*req* specifies the data that should be calculated by the callback function. This can be any combination of

- EPHEM\_TRUEPOS (true body position)
- EPHEM\_TRUEVEL (true body velocity)
- EPHEM\_BARYPOS (barycentric position)
- EPHEM\_BARYVEL (barycentric velocity)

where the barycentre refers to the system consisting of the body itself and all its children (e.g. moons). *ret* is a pointer to an array of 12 doubles, to which the function should write its results:

- *ret[0-2]*: true position (if requested)
- *ret[3-5]*: true velocity (if requested)
- *ret[6-8]*: barycentric position (if requested)
- *ret[9-11]*: barycentric velocity (if requested)

Data can be returned in either polar or cartesian format. In cartesian format, the position data blocks should contain x,y and z position (in meters), and the velocity data blocks should contain dx/dt, dy/dt and dz/dt (in m/s), where x points to the vernal equinox, y points to ecliptic zenith, and z is orthogonal to both.

In polar format, the position data blocks should contain longitude j [rad], latitude q [rad] and radial distance r [AU], and the velocity data blocks should contain dj/dt [rad/s], dq/dt [rad/s] and d r/dt [AU/s]. When returning data in polar format, include the EPHEM\_POLAR flag in the return value.

The return value should contain the flags for the data that were actually computed. For example, if both true and barycentric data were requested, but the module can only compute true positions, it should return EPHEM\_TRUEPOS | EPHEM\_TRUEVEL.

If the true and barycentric positions are identical (that is, if the body has no child objects) the return value should contain the additional flag EPHEM\_BARYISTRUE.

If both true and barycentric data are requested, but are computationally expensive to compute (for example, if they require two separate series evaluations), the module can return true positions only. Orbiter will then calculate the barycentric data directly, after evaluating the child object positions.

If a request can't be satisfied at all (e.g. if barycentric data were requested, but the module can only compute true positions), the module should calculate whatever data it can, and signal so via the return value. Orbiter will then try to convert these data to the required ones.

If the returned ephemerides are computed in terms of the barycentre of the parent body's system, the return value should include the EPHEM\_PARENTBARY flag. If the ephemerides are computed in terms of the parent body's true position, this flag should not be included.

This function is not called by Orbiter to update the planet's position during the normal simulation frame update. (For that purpose, [clbkFastEphemeris\(\)](#) is called instead). [clbkEphemeris\(\)](#) is only called if the planet state at some arbitrary time point is required, e.g. by an instrument calculating a transfer orbit.

---

**16.10.2.5 virtual int CELBODY::clbkFastEphemeris (double *simt*, int *req*, double \* *ret*) [virtual]**

Called by Orbiter to update the body's state to the next simulation frame.

**Parameters:**

*simt* simulation time (seconds)  
*req* data request bitflags (see notes)  
*ret* pointer to result vector

**Returns:**

bitflags describing returned data (see notes)

**Default action:**

None, returning 0

**Note:**

This function should perform the same function as [clbkEphemeris\(\)](#), but it will be called at each simulation frame. This means that the sampling times will be incremented in small steps, allowing for a potentially more efficient implementation, e.g. by using an interpolation scheme.

If possible, a full evaluation of a long series of perturbation terms should be avoided here, to avoid performance hits.

Note that the time parameter is passed in the form of simulation time (seconds) unlike [clbkEphemeris\(\)](#), which uses absolute MJD time. This avoids rounding errors in the time variable, and allows higher temporal resolutions.

---

**16.10.2.6 virtual bool CELBODY::clbkAtmParam (double *alt*, ATMPARAM \* *prm*) [virtual]**

Called by Orbiter to obtain atmospheric parameters at a given altitude.

**Parameters:**

*alt* altitude over planet mean radius  
*prm* pointer to [ATMPARAM](#) structure receiving results

**Returns:**

*true* if parameters have been retrieved sucessfully, *false* to indicate that the planet has no atmosphere, or if alt is above the cutoff limit for atmospheric calculations.

**Default action**

None, returning false.

**Note:**

The [ATMPARAM](#) structure contains the following fields:

```
typedef struct {
    double T;           // temperature [K]
    double p;           // pressure [Pa]
    double rho;         // density [kg/m3]
} ATMPARAM;
```

Currently, atmospheric parameters are assumed to be functions of altitude only. Local variations ("weather") are not yet supported.

The documentation for this class was generated from the following file:

- Orbitersdk/include/CelBodyAPI.h

## 16.11 CELBODY2 Class Reference

```
#include <CelBodyAPI.h>
```

Inheritance diagram for CELBODY2:

Collaboration diagram for CELBODY2:

### 16.11.1 Detailed Description

Extension to [CELBODY](#) class.

This class introduces extended atmosphere support. It contains an [ATMOSPHERE](#) class instance which handles all atmosphere data requests. The atmosphere class can be either defined directly in the celestial body's plugin module, or it can be loaded from an external module. This latter option allows to replace atmospheric models easily, without having to re-implement other parts of the code, such as the ephemeris calculations.

**See also:**

[CELBODY](#), [ATMOSPHERE](#)

### Public Member Functions

- [CELBODY2 \(OBJHANDLE hCBody\)](#)

*Constructor: Creates a [CELBODY2](#) instance for a celestial body.*

- virtual ~**CELBODY2** ()
 

*Destructor: Destroys the **CELBODY2** instance.*
- virtual void **clbkInit** (**FILEHANDLE** cfg)
 

*Module initialisation from configuration file settings.*
- **OBJHANDLE GetHandle** () const
 

*Returns the handle of the associated object.*
- **OBJHANDLE GetParent** () const
 

*Returns the handle for the parent body in the solar system hierarchy.*
- **OBJHANDLE GetChild** (DWORD idx) const
 

*Returns for a child body in the solar system hierarchy.*
- double **SidRotPeriod** () const
 

*Returns the siderial period of the celestial body.*
- **ATMOSPHERE \* GetAtmosphere** () const
 

*Returns the body's atmosphere instance.*
- virtual bool **LegacyAtmosphereInterface** () const
 

*Flags the atmosphere interface version.*

### Protected Member Functions

- void **SetAtmosphere** (**ATMOSPHERE** \*a)
 

*Assigns an atmosphere object for the celestial body.*
- bool **FreeAtmosphere** ()
 

*Remove the atmosphere instance.*
- bool **LoadAtmosphereModule** (const char \*fname)
 

*Loads an atmosphere instance from a DLL plugin.*
- bool **FreeAtmosphereModule** ()
 

*Unload the current atmosphere module.*

### Protected Attributes

- **OBJHANDLE hBody**

*handle for the associated celestial body*
- **ATMOSPHERE \* atm**

*pointer to atmosphere object*
- **HINSTANCE hAtmModule**

*library handle for external atmosphere module*

**Friends**

- class [ATMOSPHERE](#)

**16.11.2 Constructor & Destructor Documentation****16.11.2.1 CELBODY2::CELBODY2 (OBJHANDLE *hCBody*)**

Constructor. Creates a [CELBODY2](#) instance for a celestial body.

**Parameters:**

*hCBody* body handle

**16.11.2.2 virtual CELBODY2::~CELBODY2 () [virtual]**

Destructor. Destroys the [CELBODY2](#) instance.

**Default action:**

Calls the [FreeAtmosphere](#) method, to delete the atmosphere instance and unload any external atmosphere modules.

**16.11.3 Member Function Documentation****16.11.3.1 virtual void CELBODY2::clbkInit (FILEHANDLE *cfg*) [virtual]**

Module initialisation from configuration file settings.

**Parameters:**

*cfg* file handle for configuration file

**Default action:**

- Calls the base class [CELBODY::clbkInit](#) method
  - If an atmosphere module is not already loaded, and if the configuration file contains a [MODULE\\_ATM](#) entry, the [LoadAtmosphereModule](#) method is called with the corresponding module file name.

Reimplemented from [CELBODY](#).

**16.11.3.2 OBJHANDLE CELBODY2::GetParent () const**

Returns the handle for the parent body in the solar system hierarchy.

**Returns:**

Parent body handle, or NULL if no parent.

**Note:**

For primary planets, this method returns a handle to the central star. For moons, it returns a handle to the parent planet. For the central star itself, it returns NULL.

**16.11.3.3 OBJHANDLE CELBODY2::GetChild (DWORD *idx*) const**

Returns for a child body in the solar system hierarchy.

**Parameters:**

*idx* child body index ( $\geq 0$ )

**Returns:**

Child body handle, or NULL if not available.

**Note:**

For the central star, this returns the handles of the primary planets.

For planets, it returns the handles of the moons.

If *idx*  $\geq$  number of children, the function returns NULL.

**16.11.3.4 double CELBODY2::SidRotPeriod () const**

Returns the sidereal period of the celestial body.

**Returns:**

Siderial rotation period [s]

**16.11.3.5 ATMOSPHERE\* CELBODY2::GetAtmosphere () const [inline]**

Returns the body's atmosphere instance.

**Returns:**

pointer to atmosphere object, or NULL if the body has no atmosphere.

**Note:**

To provide an atmosphere for the body, the [CELBODY2](#) object should instantiate the atm member as an object of a derived [ATMOSPHERE](#) class.

**16.11.3.6 virtual bool CELBODY2::LegacyAtmosphereInterface () const [inline, virtual]**

Flags the atmosphere interface version.

**Returns:**

*false* indicates that Orbiter should use the [ATMOSPHERE](#) object returned by [GetAtmosphere](#) to query atmospheric parameters. *true* indicates that Orbiter should use the [CELBODY2::clbkAtmParam](#) method instead.

**Note:**

If the body does not have an atmosphere, this method should return *false*, and [GetAtmosphere](#) should return *NULL*.

**See also:**

[GetAtmosphere](#), [CELBODY2::clbkAtmParam](#)

**16.11.3.7 void CELBODY2::SetAtmosphere (ATMOSPHERE \* *a*) [protected]**

Assigns an atmosphere object for the celestial body.

**Parameters:**

*a* pointer to [ATMOSPHERE](#) object

**Note:**

Any previously defined atmosphere object is deallocated and replaced.

*a* = NULL will eliminate the body's atmosphere.

By default (prior to the first call to SetAtmosphere, a celestial body does not have an atmosphere.

Use this function if the atmosphere class is defined directly in the celestial body's module. For example,

```
class MyAtmosphere: public ATMOSPHERE
{
    MyAtmosphere(CELBODY2 *body) : ATMOSPHERE(body)
    {}
    ...
};

class MyCelbody: public CELBODY2
{
    MyCelbody(OBJHANDLE body) : CELBODY2 (body)
    {
        SetAtmosphere (new MyAtmosphere(this));
        ...
    }
    ...
};
```

If the atmosphere class is defined in an external module, use the [LoadAtmosphereModule](#) method instead.

**16.11.3.8 bool CELBODY2::FreeAtmosphere () [protected]**

Remove the atmosphere instance.

**Returns:**

*true* on success, *false* on failure (no atmosphere defined).

**Note:**

This method calls [FreeAtmosphereModule](#), if an external atmosphere module is loaded. Otherwise, is just deletes the atm instance.

**16.11.3.9 bool CELBODY2::LoadAtmosphereModule (const char \* *fname*) [protected]**

Loads an atmosphere instance from a DLL plugin.

**Parameters:**

*fname* DLL file name (excluding '.dll' extension and relative to 'Modules\' folder)

**Returns:**

*true* if atmosphere module could be loaded, *false* otherwise

**Note:**

If successful, this method sets the hAtmModule member to the atmospheric module instance handle, and sets the atm member by calling the CreateAtmosphere function in the module. The CreateAtmosphere function has the following interface:

```
ATMOSPHERE *CreateAtmosphere (CELBODY2 *cbody);
```

**16.11.3.10 bool CELBODY2::FreeAtmosphereModule () [protected]**

Unload the current atmosphere module.

**Returns:**

*true* indicates success, *false* indicates failure (no module loaded)

**Note:**

Before unloading the module, this function first deletes the atmosphere instance by calling the module's DeleteAtmosphere function. The interface is

```
void DeleteAtmosphere (ATMOSPHERE *atm);
```

If this function is not found in the module, the atmosphere instance is deleted directly.

The documentation for this class was generated from the following file:

- Orbitersdk/include/CelBodyAPI.h

**16.12 COLOUR4 Struct Reference**

```
#include <OrbiterAPI.h>
```

**16.12.1 Detailed Description**

colour definition

**Public Attributes**

- float **r**  
*read colour component [0..1]*
- float **g**  
*green colour component [0..1]*
- float **b**  
*blue colour component [0..1]*
- float **a**  
*alpha (opacity) component (0..1)*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/OrbiterAPI.h

## 16.13 oapi::DrawingTool Class Reference

```
#include <DrawAPI.h>
```

Inheritance diagram for oapi::DrawingTool:

### 16.13.1 Detailed Description

Base class for various 2-D drawing resources (fonts, pens, brushes, etc.).

#### Public Member Functions

- [DrawingTool \(\)](#)  
*Drawing tool constructor.*
- virtual [~DrawingTool \(\)](#)  
*Drawing tool destructor.*

The documentation for this class was generated from the following file:

- OrbiterSDK/include/[DrawAPI.h](#)

## 16.14 ELEMENTS Struct Reference

```
#include <OrbiterAPI.h>
```

### 16.14.1 Detailed Description

Kepler orbital elements.

A set of 6 scalar parameters defining the state of an object in a 2-body (Keplerian) orbit. The orbital trajectory is a conic section, either closed (circular, elliptic), or open (parabolic, hyperbolic).

#### Note:

semi-major axis  $a$  is positive for closed orbits, and negative for open orbits (in that case,  $a$  is referred to as real semi-axis).

eccentricity  $e$ :

- circular orbit:  $e = 0$
- elliptic orbit:  $0 < e < 1$
- parabolic orbit:  $e = 1$
- hyperbolic orbit:  $e > 1$

The a and e parameters define the shape of the orbit, the i, theta and omegab parameters define the orientation of the orbital plane in space, and the L parameter defines the object position along the trajectory at a given time.

This is a generic data format. Additional data are required to fully define an object's state in space (position and velocity vectors). These include the position of the orbited body, the orientation of the reference coordinate system, and the date to which the mean longitude parameter refers.

#### See also:

[ORBITPARAM](#), [Basics of orbital mechanics](#)

#### Public Attributes

- double **a**  
*semi-major axis [m]*
- double **e**  
*eccentricity*
- double **i**  
*inclination [rad]*
- double **theta**  
*longitude of ascending node [rad]*
- double **omegab**  
*longitude of periapsis [rad]*
- double **L**  
*mean longitude at epoch*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 16.15 ENGINESTATUS Struct Reference

```
#include <OrbiterAPI.h>
```

### 16.15.1 Detailed Description

Engine status.

#### Public Attributes

- double **main**  
*-1 (full retro) .. +1 (full main)*

- double **hover**  
*0 .. +1 (full hover)*
- int **attmode**  
*0=rotation, 1=translation*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 16.16 EXHAUSTSPEC Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for EXHAUSTSPEC:

### 16.16.1 Detailed Description

Engine exhaust render parameters.

See also:

[VESSEL::AddExhaust\(EXHAUSTSPEC\\*\)](#)

#### Public Attributes

- **THRUSTER\_HANDLE th**  
*handle of associated thruster (or NULL if none)*
- double \* **level**  
*pointer to variable containing exhaust level (0..1)*
- **VECTOR3 \* lpos**  
*pointer to exhaust position vector [m]*
- **VECTOR3 \* ldir**  
*pointer to engine thrust direction (=negative exhaust direction)*
- double **lsize**  
*exhaust length [m]*

- double **wsize**  
*exhaust width [m]*
- double **lofs**  
*longitudinal offset from engine [m]*
- double **modulate**  
*magnitude of random intensity variations (0..1)*
- SURFHANDLE **tex**  
*custom texture handle*
- DWORD **flags**  
*Bit flags (see [Bitflags for EXHAUSTSPEC flags field](#)).*
- UINT **id**  
*reserved*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 16.17 ExternMFD Class Reference

```
#include <MFDAPI.h>
```

### 16.17.1 Detailed Description

**ExternMFD** provides support for defining an **MFD** display in a plugin module.

**ExternMFD** provides support for defining an **MFD** display in a plugin module, e.g. for displaying the **MFD** in a dialog box. Unlike the **MFD** class described above, which defines a logical **MFD** mode, this class represents an actual **MFD** instrument, i.e. the physical display and associated push buttons.

A plugin module should derive its own **MFD** class from **ExternMFD** and overload the virtual notification callback methods.

The class interface is defined in Orbitersdk\include\[MFDAPI.h](#).

For an example using the **ExternMFD** class, see project Orbitersdk\samples\ExtMFD.

### Public Member Functions

- **ExternMFD** (const MFDSPEC &spec)  
*Constructor. Creates a new instance of **ExternMFD**.*
- virtual ~**ExternMFD** ()  
*Destructor. Deallocates the **ExternMFD** instance.*
- UINT **Id** () const

*Returns an identifier for the **MFD** instance.*

- bool **Active** () const  
*Returns a flag indicating active/passive **MFD** state.*
- **OBJHANDLE GetVessel** () const  
*Returns the handle of the vessel associated with the **MFD**.*
- virtual void **SetVessel** (**OBJHANDLE** hV)  
*Attaches the **MFD** to a different vessel.*
- **SURFHANDLE GetDisplaySurface** () const  
*Returns a handle to the surface containing the current **MFD** display.*
- const char \* **GetButtonLabel** (int bt) const  
*Returns the label currently associated with one of the **MFD** buttons.*
- bool **ProcessButton** (int bt, int event)
- bool **SendKey** (DWORD key)
- bool **Resize** (const MFDSPEC &spec)
- bool **SetMode** (int mode)
- bool **OpenModeHelp** () const
- virtual void **clbkUpdate** ()
- virtual void **clbkRefreshDisplay** (**SURFHANDLE** hSurf)
- virtual void **clbkRefreshButtons** ()
- virtual void **clbkFocusChanged** (**OBJHANDLE** hFocus)

### Public Attributes

- Instrument \* **instr**

### Protected Attributes

- **OBJHANDLE hVessel**  
*vessel associated with the **MFD***
- int **DW**
- int **DH**  
*display width, height (pixel)*
- int **pmode**  
*previous mode identifier*
- int **nbt1**
- int **nbt2**  
*number of left, right buttons*
- int **bty0**
- int **btdy**

*geometry parameters*

- int **btpressed**  
*currently pressed button (-1 if none)*

## 16.17.2 Constructor & Destructor Documentation

### 16.17.2.1 ExternMFD::ExternMFD (const MFDSPEC & *spec*)

Constructor. Creates a new instance of [ExternMFD](#).

**Parameters:**

*spec* structure containing [MFD](#) layout geometry data

**Note:**

To use a new [MFD](#) instance, it must be registered with Orbiter via a call to `oapiRegisterExternMFD()`, e.g. with `oapiRegisterExternMFD(new ExternMFD (spec))`;

To unregister an [MFD](#) instance, use `oapiUnregisterExternMFD()`. Note that `oapiUnregisterExternMFD()` automatically calls the [`~ExternMFD\(\)`](#) destructor, so the plugin should not try to delete the [MFD](#) instance manually.

### 16.17.2.2 virtual ExternMFD::~ExternMFD () [virtual]

Destructor. Deallocates the [ExternMFD](#) instance.

**Note:**

The destructor should not be called directly by the module. Instead, a call to `oapiUnregisterExternMFD()` will invoke the [`~ExternMFD\(\)`](#) destructor (or the overloaded destructor of a derived class), as well as remove the [MFD](#) instance from Orbiter's internal list of MFDs.

## 16.17.3 Member Function Documentation

### 16.17.3.1 UINT ExternMFD::Id () const

Returns an identifier for the [MFD](#) instance.

**Returns:**

A unique identifier for the [MFD](#) instance.

**Note:**

Unlike the internal [MFD](#) instances (e.g. MFDs embedded in panels) whose identifiers are in the range 0 ... MAXMFD-1, the [ExternMFD](#) class simply uses its own instance pointer (UINT)`this` to create an identifier.

### 16.17.3.2 bool ExternMFD::Active () const

Returns a flag indicating active/passive [MFD](#) state.

**Returns:**

`true` indicates that the [MFD](#) is active (switched on), `false` indicates inactive (switched off).

**16.17.3.3 OBJHANDLE ExternMFD::GetVessel () const**

Returns the handle of the vessel associated with the [MFD](#).

**Returns:**

Vessel handle associated with the [MFD](#).

**Note:**

Normally, the [ExternMFD](#) class always connects to the "focus vessel", i.e. the vessel receiving user input. If the user switches to a different vessel (e.g. via F3), then [ExternMFD](#) re-attaches itself to the new vessel.

This behaviour can be changed by overloading the [clbkFocusChanged\(\)](#) method. For example, the [MFD](#) could be forced to stick to a given vessel, regardless of the focus object.

**16.17.3.4 virtual void ExternMFD::SetVessel (OBJHANDLE *hV*) [virtual]**

Attaches the [MFD](#) to a different vessel.

**Parameters:**

*hV* vessel handle

**Default behaviour:** Sets the vessel reference to *hV*. If an [MFD](#) mode is active, the mode is closed and reopened with the new vessel reference.

**16.17.3.5 SURFHANDLE ExternMFD::GetDisplaySurface () const**

Returns a handle to the surface containing the current [MFD](#) display.

**Returns:**

Handle to the [MFD](#) display surface.

**Note:**

The handle can be used to modify or copy the current contents of the [MFD](#) display. For example, you can obtain a GDI drawing device context for the surface with [oapiGetDC\(\)](#).

**16.17.3.6 const char\* ExternMFD::GetButtonLabel (int *bt*) const**

Returns the label currently associated with one of the [MFD](#) buttons.

**Parameters:**

*bt* button number ( $0 \leq bt < \text{nbuttons}$ )

**Returns:**

Pointer to the label associated with the button (up to 3 characters, zeroterminated), or NULL if no function is associated with the button by the current [MFD](#) mode.

**Note:**

The number of buttons provided by the [MFD](#) depends on the data passed to the constructor in the MFDSPEC structure.

The module can use this method to update its button labels within the [clbkRefreshButtons\(\)](#) callback function.

**16.17.3.7 bool ExternMFD::ProcessButton (int *bt*, int *event*)**

**TODO**

**16.17.3.8 bool ExternMFD::SendKey (DWORD *key*)**

**TODO**

**16.17.3.9 bool ExternMFD::Resize (const MFDSPEC & *spec*)**

**TODO**

**16.17.3.10 bool ExternMFD::SetMode (int *mode*)**

**TODO**

**16.17.3.11 bool ExternMFD::OpenModeHelp () const**

**TODO**

**16.17.3.12 virtual void ExternMFD::clbkUpdate () [virtual]**

**TODO**

**16.17.3.13 virtual void ExternMFD::clbkRefreshDisplay (SURFHANDLE *hSurf*) [virtual]**

**TODO**

**16.17.3.14 virtual void ExternMFD::clbkRefreshButtons () [virtual]**

**TODO**

**16.17.3.15 virtual void ExternMFD::clbkFocusChanged (OBJHANDLE *hFocus*) [virtual]**

**TODO**

The documentation for this class was generated from the following file:

- Orbitersdk/include/MFDAPIL.h

## 16.18 FogParam Struct Reference

```
#include <GraphicsAPI.h>
```

Collaboration diagram for FogParam:

### 16.18.1 Detailed Description

Distance fog render parameters.

#### Public Attributes

- double `dens_0`  
*fog density at ground level*
- double `dens_ref`  
*fog density at reference altitude*
- double `alt_ref`  
*reference altitude [m]*
- `VECTOR3 col`  
*fog colour*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/GraphicsAPI.h

## 16.19 oapi::Font Class Reference

```
#include <DrawAPI.h>
```

Inheritance diagram for oapi::Font:

Collaboration diagram for oapi::Font:

### 16.19.1 Detailed Description

A font resource for drawing text. A font has a defined size, typeface, slant, weight, etc. Fonts can be selected into a [Sketchpad](#) and then apply to all subsequent Text calls.

## Public Types

- enum **Style** { **NORMAL** = 0, **BOLD** = 1, **ITALIC** = 2, **UNDERLINE** = 4 }
- Font decoration style.*

## Public Member Functions

- virtual **~Font** ()  
*Font destructor.*
- virtual **HFONT GetGDIFont** () const  
*Return the GDI handle for the font, if available.*

## Protected Member Functions

- **Font** (int height, bool prop, const char \*face, **Style** style=NORMAL, int orientation=0)  
*Font constructor.*

## 16.19.2 Member Enumeration Documentation

### 16.19.2.1 enum oapi::Font::Style

*Font* decoration style.

See also:

`Font(int,bool,char*,Style)`

Enumerator:

**NORMAL** no decoration

**BOLD** boldface

**ITALIC** italic

**UNDERLINE** underlined

## 16.19.3 Constructor & Destructor Documentation

### 16.19.3.1 oapi::Font::Font (int *height*, bool *prop*, const char \**face*, Style *style* = NORMAL, int *orientation* = 0) [inline, protected]

*Font* constructor.

Parameters:

*height* cell or character height [pixel]

*prop* proportional/fixed width flag

*face* font face name

*style* font decoration

***orientation*** text orientation [1/10 deg]

**Note:**

If *height* > 0, it represents the font cell height. If *height* < 0, its absolute value represents the character height.

The *style* parameter can be any combination of the [Style](#) enumeration items.

Overloaded font implementations should understand at least the following generic face names: "Fixed" (fixed pitch font), "Sans" (sans-serif font, and "Serif" (serif font) and translate them to appropriate specific fonts, e.g. "Courier" or "Courier New" for "Fixed", "Helvetica" or "Arial" for "Sans", and "Times" or "Times New Roman" for "Serif".

If a font name is not recognised, the *prop* value should be checked. If *prop*=true, the default "Sans" font should be used. If false, the default "Fixed" font should be used.

#### 16.19.4 Member Function Documentation

##### 16.19.4.1 virtual HFONT oapi::Font::GetGDIFont () const [inline, virtual]

Return the GDI handle for the font, if available.

**Returns:**

GDI font handle

**Note:**

Non-GDI clients should not overload this method.

The documentation for this class was generated from the following file:

- Orbitersdk/include/[DrawAPI.h](#)

## 16.20 oapi::GraphicsClient Class Reference

```
#include <GraphicsAPI.h>
```

Inheritance diagram for oapi::GraphicsClient:

Collaboration diagram for oapi::GraphicsClient:

### 16.20.1 Detailed Description

Base class for external graphics client modules.

This class defines the interface between the graphics-less version of the Orbiter core and any external plugins providing a rendering environment for the orbiter-generated scene. The [GraphicsClient](#) base class is defined in terms of generic graphics objects (meshes, textures, etc.) Derived classes can then adapt these into specific rendering objects for a given 3-D rendering engine (DX, OGL, etc.)

#### Public Member Functions

- [`GraphicsClient \(HINSTANCE hInstance\)`](#)  
*Create a graphics object.*
- [`virtual ~GraphicsClient \(\)`](#)  
*Destroy the graphics object.*
- [`virtual bool clbkInitialise \(\)`](#)  
*Perform any one-time setup tasks.*
- [`virtual void clbkRefreshVideoData \(\)`](#)  
*Request for video configuration data.*
- [`virtual SURFHANDLE clbkLoadTexture \(const char \*fname, DWORD flags=0\)`](#)  
*Texture request.*
- [`virtual SURFHANDLE clbkLoadSurface \(const char \*fname, DWORD attrib\)`](#)  
*Load a surface from file into a surface object, and return a SURFHANDLE for it.*
- [`virtual void clbkReleaseTexture \(SURFHANDLE hTex\)`](#)  
*Texture release request.*
- [`virtual bool clbkSetMeshTexture \(DEVMESHHANDLE hMesh, DWORD texidx, SURFHANDLE tex\)`](#)  
*Replace a texture in a device-specific mesh.*
- [`virtual int clbkSetMeshMaterial \(DEVMESHHANDLE hMesh, DWORD matidx, const MATERIAL \*mat\)`](#)  
*Replace properties of an existing mesh material.*

- virtual bool `clbkSetMeshProperty` (`DEVMESHHANDLE` hMesh, `DWORD` property, `DWORD` value)  
*Set custom properties for a device-specific mesh.*
- virtual `ScreenAnnotation` \* `clbkCreateAnnotation` ()  
*Create an annotation object for displaying on-screen text.*
- virtual LRESULT `RenderWndProc` (`HWND` hWnd, `UINT` uMsg, `WPARAM` wParam, `LPARAM` lParam)  
*Render window message handler.*
- virtual `BOOL` `LaunchpadVideoWndProc` (`HWND` hWnd, `UINT` uMsg, `WPARAM` wParam, `LPARAM` lParam)  
*Message handler for 'video' tab in Orbiter Launchpad dialog.*
- `VIDEODATA` \* `GetVideoData` ()  
*Returns a pointer to the VideoData structure.*
- `DWORD` `GetPopupList` (`const HWND` \*\*hPopupWnd) const  
*returns a list of popup windows owned by the render window.*
- virtual bool `clbkFullscreenMode` () const =0  
*Fullscreen mode flag.*
- virtual void `clbkGetViewportSize` (`DWORD` \*width, `DWORD` \*height) const =0  
*Returns the dimensions of the render viewport.*
- virtual bool `clbkGetRenderParam` (`DWORD` prm, `DWORD` \*value) const =0  
*Returns a specific render parameter.*
- `const void` \* `GetConfigParam` (`DWORD` paramtype) const  
*Returns a pointer to an Orbiter configuration parameter.*
- `bool` `TexturePath` (`const char` \*fname, `char` \*path) const  
*Return the full path for a texture file.*
- `SURFHANDLE` `GetVCHUDSurface` (`const VCHUDSPEC` \*\*hudspec) const  
*Returns the surface containing the virtual cockpit HUD.*
- `SURFHANDLE` `GetMFDSurface` (`int` mfd) const  
*Returns the surface containing an `MFD` display.*
- `SURFHANDLE` `GetVCMFDSurface` (`int` mfd, `const VCMFDSPEC` \*\*mfdspec) const  
*Returns the surface containing a virtual cockpit `MFD` display.*
- `DWORD` `GetBaseTileList` (`OBJHANDLE` hBase, `const SurftileSpec` \*\*tile) const  
*Returns a list of high-res surface tile specifications for a base.*
- `void` `GetBaseStructures` (`OBJHANDLE` hBase, `MESHHANDLE` \*\*mesh\_bs, `DWORD` \*nmesh\_bs, `MESHHANDLE` \*\*mesh\_as, `DWORD` \*nmesh\_as) const

*Returns meshes for generic base objects.*

- void [GetBaseShadowGeometry](#) (OBJHANDLE hBase, MESHHANDLE \*\*mesh\_sh, double \*\*elev, DWORD \*nmesh\_sh) const

*Returns base meshes in a format that can be used for shadow projections.*

- virtual void [clbkRender2DPanel](#) (SURFHANDLE \*hSurf, MESHHANDLE hMesh, MATRIX3 \*T, bool transparent=false)

*Render an instrument panel in cockpit view as a 2D billboard.*

- DWORD [LoadStars](#) (DWORD n, StarRec \*rec)

*Load star data from Orbiter's data base file.*

- DWORD [LoadConstellationLines](#) (DWORD n, ConstRec \*rec)

*Load constellation line data from Orbiter's data base file.*

### Visual object interface

- void [RegisterVisObject](#) (OBJHANDLE hObj, VISHANDLE vis)

*Register a new visual object with Orbiter.*

- void [UnregisterVisObject](#) (OBJHANDLE hObj)

*Unregister a visual before deleting it.*

- virtual int [clbkVisEvent](#) (OBJHANDLE hObj, VISHANDLE vis, DWORD msg, UINT context)

*Message callback for a visual object.*

- virtual MESHHANDLE [clbkGetMesh](#) (VISHANDLE vis, UINT idx)

*Return a mesh handle for a visual, defined by its index.*

- virtual int [clbkEditMeshGroup](#) (DEVMESHHANDLE hMesh, DWORD grpidx, GROUPEDITSPEC \*ges)

*Mesh group editing interface for device-specific meshes.*

### Dialog interface

- virtual void [clbkPreOpenPopup](#) ()

*Popup window open notification.*

### Particle stream methods

- virtual ParticleStream \* [clbkCreateParticleStream](#) (PARTICLESTREAMSPEC \*pss)

*Create a generic particle stream.*

- virtual ParticleStream \* [clbkCreateExhaustStream](#) (PARTICLESTREAMSPEC \*pss, OBJHANDLE hVessel, const double \*lvl, const VECTOR3 \*ref, const VECTOR3 \*dir)

*Create a particle stream associated with a vessel.*

- virtual ParticleStream \* [clbkCreateExhaustStream](#) (PARTICLESTREAMSPEC \*pss, OBJHANDLE hVessel, const double \*lvl, const VECTOR3 &ref, const VECTOR3 &dir)

*Create a particle stream associated with a vessel.*

- virtual `ParticleStream * clbkCreateReentryStream (PARTICLESTREAMSPEC *pss, OBJHANDLE hVessel)`

*Create a vessel particle stream for reentry heating effect.*

### Surface-related methods

- virtual `SURFHANDLE clbkCreateSurfaceEx (DWORD w, DWORD h, DWORD attrib)`  
*Create a surface for texturing, as a blitting source, etc.*
- virtual `SURFHANDLE clbkCreateSurface (DWORD w, DWORD h, SURFHANDLE hTemplate=NULL)`  
*Create an offscreen surface.*
- virtual `SURFHANDLE clbkCreateTexture (DWORD w, DWORD h)`  
*Create a texture for rendering.*
- virtual `SURFHANDLE clbkCreateSurface (HBITMAP hBmp)`  
*Create an offscreen surface from a bitmap.*
- virtual void `clbkIncrSurfaceRef (SURFHANDLE surf)`  
*Increment the reference counter of a surface.*
- virtual bool `clbkReleaseSurface (SURFHANDLE surf)`  
*Decrement surface reference counter, release surface if counter reaches 0.*
- virtual bool `clbkGetSurfaceSize (SURFHANDLE surf, DWORD *w, DWORD *h)`  
*Return the width and height of a surface.*
- virtual bool `clbkSetSurfaceColourKey (SURFHANDLE surf, DWORD ckey)`  
*Set transparency colour key for a surface.*
- virtual DWORD `clbkGetDeviceColour (BYTE r, BYTE g, BYTE b)`  
*Convert an RGB colour triplet into a device-specific colour value.*

### Surface blitting methods

- virtual bool `clbkBlt (SURFHANDLE tgt, DWORD tgtx, DWORD tgty, SURFHANDLE src, DWORD flag=0) const`  
*Copy one surface into an area of another one.*
- virtual bool `clbkBlt (SURFHANDLE tgt, DWORD tgtx, DWORD tgty, SURFHANDLE src, DWORD srcx, DWORD srcy, DWORD w, DWORD h, DWORD flag=0) const`  
*Copy a rectangle from one surface to another.*
- virtual bool `clbkScaleBlt (SURFHANDLE tgt, DWORD tgtx, DWORD tgty, DWORD tgtw, DWORD tgth, SURFHANDLE src, DWORD srcx, DWORD srcy, DWORD srcw, DWORD srch, DWORD flag=0) const`  
*Copy a rectangle from one surface to another, stretching or shrinking as required.*
- virtual int `clbkBeginBltGroup (SURFHANDLE tgt)`  
*Begins a block of blitting operations to the same target surface.*

- virtual int `clbkEndBltGroup ()`  
*Ends a block of blitting operations to the same target surface.*
- virtual bool `clbkFillSurface (SURFHANDLE surf, DWORD col) const`  
*Fill a surface with a uniform colour.*
- virtual bool `clbkFillSurface (SURFHANDLE surf, DWORD tgtx, DWORD tgty, DWORD w, DWORD h, DWORD col) const`  
*Fill an area in a surface with a uniform colour.*
- virtual bool `clbkCopyBitmap (SURFHANDLE pdds, HBITMAP hbm, int x, int y, int dx, int dy)`  
*Copy a bitmap object into a surface.*

## 2-D drawing interface

- virtual `Sketchpad * clbkGetSketchpad (SURFHANDLE surf)`  
*Create a 2-D drawing object ("sketchpad") associated with a surface.*
- virtual void `clbkReleaseSketchpad (Sketchpad *sp)`  
*Release a drawing object.*
- virtual `Font * clbkCreateFont (int height, bool prop, const char *face, oapi::Font::Style style=oapi::Font::NORMAL, int orientation=0) const`  
*Create a font resource for 2-D drawing.*
- virtual void `clbkReleaseFont (Font *font) const`  
*De-allocate a font resource.*
- virtual `Pen * clbkCreatePen (int style, int width, DWORD col) const`  
*Create a pen resource for 2-D drawing.*
- virtual void `clbkReleasePen (Pen *pen) const`  
*De-allocate a pen resource.*
- virtual `Brush * clbkCreateBrush (DWORD col) const`  
*Create a brush resource for 2-D drawing.*
- virtual void `clbkReleaseBrush (Brush *brush) const`  
*De-allocate a brush resource.*

## GDI-related methods

- virtual `HDC clbkGetSurfaceDC (SURFHANDLE surf)`  
*Return a Windows graphics device interface handle for a surface.*
- virtual void `clbkReleaseSurfaceDC (SURFHANDLE surf, HDC hDC)`  
*Release a Windows graphics device interface.*

## Marker and label-related methods

- DWORD `GetCelestialMarkers (const LABELLIST **cm_list) const`

*Returns an array of celestial marker lists.*

- DWORD [GetSurfaceMarkers](#) (OBJHANDLE hObj, const LABELLIST \*\*sm\_list) const  
*Returns an array of surface marker lists for a planet.*

## Public Attributes

- HWND [hVid](#)  
*Window handle of Launchpad video tab, if available.*

## Protected Member Functions

- virtual bool [clbkUseLaunchpadVideoTab](#) () const  
*Launchpad video tab indicator.*
- virtual HWND [clbkCreateRenderWindow](#) ()  
*Simulation session start notification.*
- virtual void [clbkPostCreation](#) ()  
*Simulation startup finalisation.*
- virtual void [clbkCloseSession](#) (bool fastclose)  
*End of simulation session notification.*
- virtual void [clbkDestroyRenderWindow](#) (bool fastclose)  
*Render window closure notification.*
- virtual void [clbkUpdate](#) (bool running)  
*Per-frame update notification.*
- virtual void [clbkRenderScene](#) ()=0  
*Per-frame render notification.*
- virtual bool [clbkDisplayFrame](#) ()  
*Display a scene on screen after rendering it.*
- void [Render2DOverlay](#) ()  
*Notifies Orbiter to initiate rendering of the 2D scene overlay.*
- virtual void [clbkStoreMeshPersistent](#) (MESHHANDLE hMesh, const char \*fname)  
*Store a persistent mesh template.*
- void [ShowDefaultSplash](#) ()  
*Displays the default Orbiter splash screen on top of the render window.*
- HINSTANCE [ModuleInstance](#) () const  
*Returns the graphics module instance handle.*

- HINSTANCE [OrbiterInstance \(\) const](#)  
*Returns the orbiter core instance handle.*
- HWND [LaunchpadVideoTab \(\) const](#)  
*Returns the window handle of the 'video' tab of the Orbiter Launchpad dialog.*

### Protected Attributes

- [SURFHANDLE surfBltTgt](#)  
*target surface for a blitting group (-I=none, NULL=main window render surface)*

### Friends

- class [::Orbiter](#)  
*Orbiter private class.*

### Classes

- struct [LABELLIST](#)  
*Label list description for celestial and surface markers.*
- struct [VIDEODATA](#)  
*Structure containing default video options, as stored in Orbiter.cfg.*

## 16.20.2 Constructor & Destructor Documentation

### 16.20.2.1 oapi::GraphicsClient::GraphicsClient (HINSTANCE *hInstance*)

Create a graphics object.

The graphics object is typically created during module initialisation (see [InitModule](#)). Once the client is created, it must be registered with the Orbiter core via the [oapiRegisterGraphicsClient](#) function.

#### Parameters:

*hInstance* module instance handle (as passed to [InitModule](#))

### 16.20.2.2 virtual oapi::GraphicsClient::~GraphicsClient () [virtual]

Destroy the graphics object.

Usually, the graphics object is destroyed when the module is unloaded (see [opcDLLEXIT](#)), after it has been detached from the Orbiter core via a call to [oapiUnregisterGraphicsClient](#).

### 16.20.3 Member Function Documentation

#### 16.20.3.1 virtual bool oapi::GraphicsClient::clbkInitialise () [virtual]

Perform any one-time setup tasks.

This includes enumerating drivers, graphics modes, etc. Derived classes should also call the base class method to allow default setup.

##### Default action:

Initialises the VideoData structure from the Orbiter.cfg file

##### Calling sequence:

Called during processing of oapiRegisterGraphicsClient, after the Launchpad Video tab has been inserted (if clbkUseLaunchpadVideoTab returns true).

#### 16.20.3.2 virtual void oapi::GraphicsClient::clbkRefreshVideoData () [inline, virtual]

Request for video configuration data.

Called by Orbiter before the render window is opened or configuration parameters are written to file. Applications should here either update the provided **VIDEODATA** structure from any user selections made in the Launchpad Video tab and leave it to Orbiter to write these parameters to Orbiter.cfg, or write the current video settings to their own configuration file.

##### Default action:

None.

#### 16.20.3.3 virtual SURFHANDLE oapi::GraphicsClient::clbkLoadTexture (const char \* *fname*, DWORD *flags* = 0) [inline, virtual]

Texture request.

Load a texture from a file into a device-specific texture object, and return a generic SURFHANDLE for it. Derived classes should overload this method to add texture support. Usually, the client should read Orbiter's default texture files (in DXT? format). However, the client also has the option to load its own texture files stored in a different format, and pass them back via the SURFHANDLE interface.

##### Parameters:

*fname* texture file name with path relative to orbiter texture folders; can be used as input for OpenTextureFile.

*flags* request for texture properties

##### Returns:

Texture handle, cast into generic SURFHANDLE, or NULL if texture could not be loaded.

##### Default action:

Return NULL.

**Note:**

If the client loads its own of texture files, they can either be installed in the default locations, replacing Orbiter's set of textures, or stored alongside the original textures, using different names or directory locations. In the latter case, the fname parameter passed to clbkLoadTexture must be adapted accordingly (for example, by replacing the dds extension with jpg, or by adding an 'OGL/' prefix to the path name, etc). Not overwriting the original texture set has the advantage that other graphics clients relying on the original textures can still be used.

The following flags are supported:

- bit 0 set: force creation in system memory
- bit 1 set: decompress, even if format is supported by device
- bit 2 set: don't load mipmaps, even if supported by device
- bit 3 set: load as global resource (can be managed by graphics client)

If bit 3 of flags is set, orbiter will not try to modify or release the texture. The client should manage the texture (i.e. keep it in a repository and release it at destruction). Any further call of clbkLoadTexture should first scan the repository. If the texture is already present, the function should just return a pointer to it.

#### **16.20.3.4 virtual SURFHANDLE oapi::GraphicsClient::clbkLoadSurface (const char \* *fname*, DWORD *attrib*) [inline, virtual]**

Load a surface from file into a surface object, and return a SURFHANDLE for it.

**Parameters:**

*fname* texture file name with path relative to orbiter texture folders  
*attrib* [Surface and texture attributes](#) (see notes)

**Returns:**

A SURFHANDLE for the loaded surface, for example a pointer to the surface object.

**Note:**

If the request refers to a static surface that has already be loaded, or if the client buffers the unmodified surfaces after loading, it can simply return a handle to the existing surface object, instead of reading it again from file.

The attrib bitflag can contain one of the following main attributes:

- OAPISURFACE\_RO: Load the surface to be readable by the GPU pipeline
- OAPISURFACE\_RW: Load the surface to be readable and writable by the GPU pipeline
- OAPISURFACE\_GDI: Load the surface to be readable and writable by the CPU, and can be blitted into an uncompressed RO or RW surface without alpha channel
- OAPISURFACE\_STATIC: Load the surface to be readable by the GPU pipeline In addition, the flag can contain any of the following auxiliary attributes:
- OAPISURFACE\_MIPMAPS: Load the mipmaps for the surface from file, or create them if necessary
- OAPISURFACE\_NOMIPMAPS: Don't load mipmaps, even if they are available in the file
- OAPISURFACE\_NOALPHA: Load the surface without an alpha channel
- OAPISURFACE\_UNCOMPRESS: Uncompress the surface on loading.

**See also:**

[oapiCreateSurface\(DWORD,DWORD,DWORD\)](#)

**16.20.3.5 virtual void oapi::GraphicsClient::clbkReleaseTexture (SURFHANDLE hTex) [inline, virtual]**

Texture release request.

Called by Orbiter when a previously loaded texture can be released from memory. The client can use the appropriate device-specific method to release the texture.

**Parameters:**

*hTex* texture handle

**Default action:**

None.

**16.20.3.6 virtual bool oapi::GraphicsClient::clbkSetMeshTexture (DEVMESHHANDLE hMesh, DWORD texidx, SURFHANDLE tex) [inline, virtual]**

Replace a texture in a device-specific mesh.

**Parameters:**

*hMesh* device mesh handle

*texidx* texture index ( $\geq 0$ )

*tex* texture handle

**Returns:**

Should return *true* if operation successful, *false* otherwise.

**Default action:**

None, returns *false*.

**16.20.3.7 virtual int oapi::GraphicsClient::clbkSetMeshMaterial (DEVMESHHANDLE hMesh, DWORD matidx, const MATERIAL \*mat) [inline, virtual]**

Replace properties of an existing mesh material.

**Parameters:**

*hMesh* device mesh handle

*matidx* material index ( $\geq 0$ )

*mat* pointer to material structure

**Returns:**

Overloaded functions should return an integer error flag, with the following codes: 0="success", 3="invalid mesh handle", 4="material index out of range"

**Default action:**

, None, returns 2 ("client does not support operation").

**16.20.3.8 virtual bool oapi::GraphicsClient::clbkSetMeshProperty (DEVMESHHANDLE *hMesh*, DWORD *property*, DWORD *value*) [inline, virtual]**

Set custom properties for a device-specific mesh.

**Parameters:**

*hMesh* device mesh handle  
*property* property tag  
*value* new mesh property value

**Returns:**

The method should return *true* if the property tag was recognised and the request could be executed, *false* otherwise.

**Note:**

Currently only a single mesh property request type will be sent, but this may be extended in future versions:

- MESHPROPERTY\_MODULATEMATALPHA
  - if value==0 (default) disable material alpha information in textured mesh groups (only use texture alpha channel).
  - if value<>0 modulate (mix) material alpha values with texture alpha maps.

**Default action:**

None, returns *false*.

**16.20.3.9 void oapi::GraphicsClient::RegisterVisObject (OBJHANDLE *hObj*, VISHANDLE *vis*)**

Register a new visual object with Orbiter.

**Parameters:**

*hObj* handle of the object to register the visual with  
*vis* identifier for the visual (passed to the message callback function)

**Note:**

When the client creates a visual for an orbiter object (such as vessels and planets), it must register them with the core by calling RegisterVisObject. This will allow the visual to receive event notifications via clbkVisEvent.

Visuals should not be persistent, but should be created when an object comes into visual range of an observer camera, and deleted when the object moves out of visual range.

If a client supports multiple views, it should not register visuals for an object in each view, but only once when the object is rendered in any of the views, and unregister when the object is no longer rendered in any of the views.

*vis* should be a nonzero handle that allows the client to uniquely identify the visual (e.g. a pointer to a client-specific visual object instance). The handle is passed to the clbkVisEvent method, and also to any [VESSEL](#) methods that use VISHANDLES.

For vessel visuals, RegisterVisObject will trigger a [VESSEL2::clbkVisualCreated](#) notification to the vessel module, if it exists.

**See also:**

[UnregisterVisObject](#), [clbkVisEvent](#), [clbkVisualCreated](#)

**16.20.3.10 void oapi::GraphicsClient::UnregisterVisObject (OBJHANDLE *hObj*)**

Unregister a visual before deleting it.

**Parameters:**

*hObj* handle of the object for which the visual is un-registered.

**Note:**

Before the client deletes a visual (e.g. when it runs out of the camera visual range) it must unregister it from the core.

Once the visual is un-registered, Orbiter will no longer generate visual events via clbkVisEvent for it. For vessel visuals, UnregisterVisObject will trigger a [VESSEL2::clbkVisualDestroyed](#) notification to the vessel module, if it exists.

**See also:**

[RegisterVisObject](#), [clbkVisEvent](#)

**16.20.3.11 virtual int oapi::GraphicsClient::clbkVisEvent (OBJHANDLE *hObj*, VISHANDLE *vis*, DWORD *msg*, UINT *context*) [virtual]**

Message callback for a visual object.

**Parameters:**

*hObj* handle of the object that created the message

*vis* client-supplied identifier for the visual

*msg* event identifier

*context* message context

**Returns:**

Function should return 1 if it processes the message, 0 otherwise.

**Default action:**

None, returns 0.

**Note:**

Messages are generated by Orbiter for objects that have been registered with [RegisterVisObject](#) by the client, until they are un-registered with [UnregisterVisObject](#).

Currently only vessel objects create visual messages.

For currently supported event types, see [Identifiers for visual events](#).

The *vis* pointer passed to this function is the same as that provided by RegisterVisObject. It can be used by the client to identify the visual object for which the message was created.

**See also:**

[RegisterVisObject](#), [UnregisterVisObject](#), [Identifiers for visual events](#)

**16.20.3.12 virtual MESHHANDLE oapi::GraphicsClient::clbkGetMesh (VISHANDLE *vis*, UINT *idx*) [inline, virtual]**

Return a mesh handle for a visual, defined by its index.

**Parameters:**

*vis* visual identifier

*idx* mesh index ( $\geq 0$ )

**Returns:**

Mesh handle (client-specific)

**Note:**

Derived clients should return a handle that identifies a mesh for the visual (in client-specific format). Orbiter calls this method in response to a [VESSEL::GetMesh](#) call by an vessel module.

**16.20.3.13 virtual int oapi::GraphicsClient::clbkEditMeshGroup (DEVMESHHANDLE *hMesh*, DWORD *grpidx*, GROUPEDITSPEC \**ges*) [inline, virtual]**

Mesh group editing interface for device-specific meshes.

**Parameters:**

*hMesh* device mesh handle

*grpidx* mesh group index ( $\geq 0$ )

*ges* mesh group modification specs

**Returns:**

Should return 0 on success, or error flags  $> 0$ .

**Default action:**

None, returns -1.

**Note:**

Clients should implement this method to allow the modification of individual groups in a device-specific mesh. Modifications may include vertex values, index lists, texture and material indices, and user flags.

**16.20.3.14 virtual void oapi::GraphicsClient::clbkPreOpenPopup () [inline, virtual]**

Popup window open notification.

**Note:**

This method is called just before a popup window (e.g. dialog box) is opened. It allows the client to prepare for subsequent rendering of the window, if necessary.

**16.20.3.15 virtual ParticleStream\* oapi::GraphicsClient::clbkCreateParticleStream (PARTICLESTREAMSPEC \**pss*) [virtual]**

Create a generic particle stream.

**Parameters:**

*pss* particle stream parameters

**Returns:**

Pointer to new particle stream.

**Default action:**

None, returns NULL. Derived classes should overload this method to return a ParticleStream-derived class instance in order to support particle streams.

**See also:**

[ParticleStream](#)

**16.20.3.16 virtual ParticleStream\* oapi::GraphicsClient::clbkCreateExhaustStream (PARTICLESTREAMSPEC \**pss*, OBJHANDLE *hVessel*, const double \**lvl*, const VECTOR3 \**ref*, const VECTOR3 \**dir*) [virtual]**

Create a particle stream associated with a vessel.

Typically used for exhaust and plasma effects, but can also be used for other types of particles.

**Parameters:**

*pss* particle stream parameters

*hVessel* vessel handle

*lvl* pointer to exhaust level control variable

*ref* pointer to stream source position (vessel frame) [**m**]

*dir* pointer to stream direction (vessel frame)

**Returns:**

Pointer to new particle stream

**Default action:**

None, returns NULL. Derived classes should overload this method to return a ParticleStream-derived class instance in order to support exhaust streams.

**Note:**

The lvl, ref and dir parameters may be modified by orbiter after the stream has been created, e.g. to reflect changes in engine thrust level or gimballing.

**16.20.3.17 virtual ParticleStream\* oapi::GraphicsClient::clbkCreateExhaustStream (PARTICLESTREAMSPEC \**pss*, OBJHANDLE *hVessel*, const double \**lvl*, const VECTOR3 & *ref*, const VECTOR3 & *dir*) [virtual]**

Create a particle stream associated with a vessel.

Typically used for exhaust and plasma effects, but can also be used for other types of particles.

**Parameters:**

*pss* particle stream parameters  
*hVessel* vessel handle  
*lvl* pointer to exhaust level control variable  
*ref* pointer to stream source position (vessel frame) [**m**]  
*dir* pointer to stream direction (vessel frame)

**Returns:**

Pointer to new particle stream

**Default action:**

None, returns NULL. Derived classes should overload this method to return a ParticleStream-derived class instance in order to support exhaust streams.

**Note:**

The *lvl* parameter may be modified by orbiter after the stream has been created, e.g. to reflect changes in engine thrust level.

The *ref* and *dir* parameters are fixed in this version of the method.

**16.20.3.18 virtual ParticleStream\* oapi::GraphicsClient::clbkCreateReentryStream (PARTICLESTREAMSPEC \**pss*, OBJHANDLE *hVessel*) [virtual]**

Create a vessel particle stream for reentry heating effect.

**Parameters:**

*pss* particle stream parameters  
*hVessel* vessel handle

**Returns:**

Pointer to new particle stream

**Default action:**

None, returns NULL. Derived classes should overload this method to return a ParticleStream-derived class instance in order to support reentry streams.

**16.20.3.19 virtual ScreenAnnotation\* oapi::GraphicsClient::clbkCreateAnnotation 0 [virtual]**

Create an annotation object for displaying on-screen text.

**Returns:**

Pointer to new screen annotation object.

**Default action:**

Dynamically allocates a 'ScreenAnnotation' instance and returns a pointer to it.

**16.20.3.20 virtual LRESULT oapi::GraphicsClient::RenderWndProc (HWND *hWnd*, UINT *uMsg*, WPARAM *wParam*, LPARAM *lParam*) [virtual]**

Render window message handler.

Derived classes should also call the base class method to allow default message processing.

**Parameters:**

*hWnd* render window handle

*uMsg* Windows message identifier

*wParam* WPARAM message parameter

*lParam* LPARAM message parameter

**Returns:**

The return value depends on the message being processed.

**Note:**

This is the standard Windows message handler for the render window.

This method currently intercepts only the WM\_CLOSE and WM\_DESTROY messages, and passes everything else to the Orbiter core message handler.

**16.20.3.21 virtual BOOL oapi::GraphicsClient::LaunchpadVideoWndProc (HWND *hWnd*, UINT *uMsg*, WPARAM *wParam*, LPARAM *lParam*) [virtual]**

Message handler for 'video' tab in Orbiter Launchpad dialog.

Overload this method to display and retrieve video parameters using the Launchpad video tab. This method acts like a standard Windows dialog message handler.

**Parameters:**

*hWnd* window handle for video tab

*uMsg* Windows message

*wParam* WPARAM message value

*lParam* LPARAM message value

**Returns:**

The return value depends on the message type and the action taken.

**Default action:**

Do nothing, return FALSE.

**16.20.3.22 VIDEODATA\* oapi::GraphicsClient::GetVideoData () [inline]**

Returns a pointer to the VideoData structure.

This structure contains the user selection for video parameters as stored in the Orbiter.cfg file. You can use this structure to retrieve and present video options to the user, or ignore it and define your own method (e.g. reading/writing to a separate config file)

**Returns:**

pointer to [VIDEODATA](#) structure containing default video settings

**16.20.3.23 DWORD oapi::GraphicsClient::GetPopupList (const HWND \*\* *hPopupWnd*) const**

returns a list of popup windows owned by the render window.

**Parameters:**

→ *hPopupWnd* on exit, points to a list of window handles

**Returns:**

Number of entries in the list.

**Note:**

The list returned by this method contains the handles of popup windows that are to be rendered on top of the render viewport (e.g. dialog boxes).

A client can use this list if it requires a special method of displaying the popup windows. Typically, this is the case in fullscreen render modes, where the dialog contents may need to be blitted manually into the render surface.

**16.20.3.24 virtual bool oapi::GraphicsClient::clbkFullscreenMode () const [pure virtual]**

Fullscreen mode flag.

**Returns:**

true if the client is set up for running in fullscreen mode, false for windowed mode.

**16.20.3.25 virtual void oapi::GraphicsClient::clbkGetViewportSize (DWORD \* *width*, DWORD \* *height*) const [pure virtual]**

Returns the dimensions of the render viewport.

**Parameters:**

*width* render viewport width [pixel]

*height* render viewport height [pixel]

**Note:**

This function is called by orbiter after the render window or fullscreen renderer has been created (see [clbkCreateRenderWindow](#)).

This should normally return the screen resolution in fullscreen mode, and the size of the render window client area in windowed mode, clients can also return smaller values if they only use part of the screen area for scene rendering.

**16.20.3.26 virtual bool oapi::GraphicsClient::clbkGetRenderParam (DWORD *prm*, DWORD \* *value*) const [pure virtual]**

Returns a specific render parameter.

**Parameters:**

← *prm* parameter identifier (see [Render parameter identifiers](#))

**See also:**

[Render parameter identifiers](#))

**Parameters:**

→ *value* value of the queried parameter

**Returns:**

true if the specified parameter is supported by the client, false if not.

**16.20.3.27 const void\* oapi::GraphicsClient::GetConfigParam (DWORD *paramtype*) const**

Returns a pointer to an Orbiter configuration parameter.

This function can be used to access various configuration parameters defined in the Orbiter core (e.g. user selections in the Launchpad dialog box).

**Parameters:**

*paramtype* Parameter identifier (see [Configuration parameter identifiers](#))

**Returns:**

Pointer to parameter

**Note:**

The pointer must be cast into the appropriate variable type. The variable types can be found in the parameter type list ([Configuration parameter identifiers](#)).

**Example:**

```
double lightscale = *(double*)GetConfigParam (CFGPRM_SURFACELIGHTBRT);
```

**16.20.3.28 bool oapi::GraphicsClient::TexturePath (const char \**fname*, char \**path*) const**

Return the full path for a texture file.

Returns the fully qualified path for texture file '*fname*' in '*path*', relative to the orbiter root directory. The search method conforms to the standard orbiter convention (first search under Textures2, then under Textures directory) Example: for *fname*="mypath\tx1.dds", this may return ".\Textures2\mypath\tx1.dds" or ".\Textures\mypath\tx1.dds" Return value is false if no file is found in either directory

**Parameters:**

*fname* texture file name (with path relative to an Orbiter texture directory)

*path* string into which the full path is copied

**Returns:**

true if file was found, false otherwise.

**16.20.3.29 SURFHANDLE oapi::GraphicsClient::GetVCHUDSurface (const VCHUDSPEC \*\*  
*hudspec*) const**

Returns the surface containing the virtual cockpit HUD.

**Parameters:**

→ *hudspec* pointer to structure containing mesh and group index, and size parameters of VC HUD object

**Returns:**

HUD surface handle, or NULL if not available

**16.20.3.30 SURFHANDLE oapi::GraphicsClient::GetMFDSurface (int *mfd*) const**

Returns the surface containing an [MFD](#) display.

**Parameters:**

*mfd* [MFD](#) identifier ( $0 \leq mfd < \text{MAXMFD}$ )

**Returns:**

[MFD](#) display surface handle, or NULL if not available

**16.20.3.31 SURFHANDLE oapi::GraphicsClient::GetVCMFDSurface (int *mfd*, const VCMFD-  
SPEC \*\**mfdspec*) const**

Returns the surface containing a virtual cockpit [MFD](#) display.

**Parameters:**

← *mfd* [MFD](#) identifier ( $0 \leq mfd < \text{MAXMFD}$ )

→ *mfdspec* pointer to structure containing mesh and group index of the VC [MFD](#) display object

**Returns:**

[MFD](#) display surface handle, or NULL if not available

**16.20.3.32 DWORD oapi::GraphicsClient::GetBaseTileList (OBJHANDLE *hBase*, const Surftile-  
Spec \*\**tile*) const**

Returns a list of high-res surface tile specifications for a base.

**Parameters:**

*hBase* surface base handle

*tile* pointer to a list of tile specifications, or NULL if none defined

**Returns:**

number of surface tiles defined for the base

**16.20.3.33 void oapi::GraphicsClient::GetBaseStructures (OBJHANDLE *hBase*, MESHHANDLE \*\**mesh\_bs*, DWORD \**nmesh\_bs*, MESHHANDLE \*\**mesh\_as*, DWORD \**nmesh\_as*) const**

Returns meshes for generic base objects.

**Parameters:**

*hBase* surface base handle

*mesh\_bs* mesh list for objects rendered before shadows (NULL if none)

*nmesh\_bs* list length of mesh\_bs list

*mesh\_as* mesh list for objects rendered after shadows (NULL if none)

*nmesh\_as* list length of mesh\_as list

**Note:**

The lists contain mesh objects as well as generic object primitives (blocks, tanks, hangars, etc.) All generic objects are separated into objects rendered before and after shadows, and compressed into one mesh each, such that all objects with the same textures are merged into a single group.

**16.20.3.34 void oapi::GraphicsClient::GetBaseShadowGeometry (OBJHANDLE *hBase*, MESHHANDLE \*\**mesh\_sh*, double \*\**elev*, DWORD \**nmesh\_sh*) const**

Returns base meshes in a format that can be used for shadow projections.

**Parameters:**

*hBase* surface base handle

*mesh\_sh* list of base object meshes

*elev* list of object elevation references [m]

*nmesh\_sh* length of mesh\_sh list

**Note:**

This method returns the mesh geometry (without textures and materials) for all mesh objects rendered *after* shadows. Unlike [GetBaseStructures\(\)](#), this does not merge mesh groups from different objects, so shadow projections can be calculated on a per-object basis (onto the local horizon plane). the *elev* list is filled with elevation offsets of each object from the reference plane of the base.

**16.20.3.35 virtual void oapi::GraphicsClient::clbkRender2DPanel (SURFHANDLE \* *hSurf*, MESHHANDLE *hMesh*, MATRIX3 \* *T*, bool *transparent* = false) [virtual]**

Render an instrument panel in cockpit view as a 2D billboard.

**Parameters:**

*hSurf* array of texture handles for the panel surface

***hMesh*** billboard mesh handle

***T*** transformation matrix for panel mesh vertices (2D)

***transparent*** If true, panel should be rendered transparent

#### Default action:

None.

#### Note:

The texture index of each group in the mesh is interpreted as index into the hSurf array. Special indices are TEXIDX\_MFD0 and above, which specify the surfaces representing the MFD displays. These are obtained separately and don't need to be present in the hSurf list.

The *transparent* flag is used when rendering the default "glass cockpit" if the user requested. "transparent MFDs". The renderer can then use e.g. additive blending for rendering the panel.

### 16.20.3.36 virtual SURFHANDLE oapi::GraphicsClient::clbkCreateSurfaceEx (DWORD *w*, DWORD *h*, DWORD *attrib*) [inline, virtual]

Create a surface for texturing, as a blitting source, etc.

Surfaces are used for offscreen bitmap and texture manipulation, blitting and rendering. Derived classes should create a device-specific surface, and return a cast to a generic Orbiter SURFHANDLE.

#### Parameters:

***w*** surface width [pixels]

***h*** surface height [pixels]

***attrib*** Surface and texture attributes (bitflags). See notes.

#### Returns:

Surface handle (in the simplest case, just a pointer to the surface, cast to a SURFHANDLE). On failure, this method should return NULL.

#### Default action:

None, returns NULL.

#### Note:

The attribute flag can contain one of the following main attributes:

- OAPISURFACE\_RO: create a surface that can be read by the GPU pipeline, and that can be updated from system memory.
- OAPISURFACE\_RW: create a surface that can be read and written by the GPU pipeline, and that can be updated from system memory.
- OAPISURFACE\_GDI: create a surface that can be read and written from the CPU, and can be blitted into an uncompressed RO or RW surface without an alpha channel In addition, the flag can contain any combination of the following auxiliary attributes:
- OAPISURFACE\_MIPMAPS: create a full chain of mipmaps for the surface if possible
- OAPISURFACE\_NOALPHA: create a surface without an alpha channel

**16.20.3.37 virtual SURFHANDLE oapi::GraphicsClient::clbkCreateSurface (DWORD *w*, DWORD *h*, SURFHANDLE *hTemplate* = NULL) [inline, virtual]**

Create an offscreen surface.

Surfaces are used for offscreen bitmap and texture manipulation, blitting and rendering. Derived classes should create a device-specific surface, and return a cast to a generic Orbiter SURFHANDLE.

**Parameters:**

*w* surface width [pixels]

*h* surface height [pixels]

*hTemplate* surface format template

**Returns:**

pointer to surface, cast into a SURFHANDLE, or NULL to indicate failure.

**Default action:**

None, returns NULL.

**Note:**

If *hTemplate* is provided, this method should create the new surface with the same pixel format.

**See also:**

[clbkCreateTexture](#), [clbkReleaseSurface](#)

**16.20.3.38 virtual SURFHANDLE oapi::GraphicsClient::clbkCreateTexture (DWORD *w*, DWORD *h*) [inline, virtual]**

Create a texture for rendering.

**Parameters:**

*w* texture width

*h* texture height

**Returns:**

pointer to texture, returned as generic SURFHANDLE. NULL indicates failure.

**Note:**

This method is similar to [clbkCreateSurface](#), but the returned surface handle must be usable as a texture when rendering the scene. Clients which don't differentiate between offscreen surfaces and textures may use identical code for both functions.

Some clients may put restrictions on the texture format (e.g. require square size (*w*=*h*), and/or powers of two (*w*= $2^n$ ). If the texture cannot be created with the requested size, this method should return NULL.

**See also:**

[clbkCreateSurface](#), [clbkReleaseSurface](#)

**16.20.3.39 virtual SURFHANDLE oapi::GraphicsClient::clbkCreateSurface (HBITMAP *hBmp*)  
[virtual]**

Create an offscreen surface from a bitmap.

**Parameters:**

*hBmp* bitmap handle

**Returns:**

surface handle, or NULL to indicate failure

**Default action:**

Creates a surface of the same size as the bitmap, and uses clbkCopyBitmap to copy the bitmap over.

**Note:**

The reference counter for the new surface is set to 1.

**See also:**

[clbkIncrSurfaceRef](#), [clbkReleaseSurface](#)

**16.20.3.40 virtual void oapi::GraphicsClient::clbkIncrSurfaceRef (SURFHANDLE *surf*)  
[inline, virtual]**

Increment the reference counter of a surface.

**Parameters:**

*surf* surface handle

**Default action:**

None.

**Note:**

Derived classes should keep track on surface references, and overload this function to increment the reference counter.

**16.20.3.41 virtual bool oapi::GraphicsClient::clbkReleaseSurface (SURFHANDLE *surf*)  
[inline, virtual]**

Decrement surface reference counter, release surface if counter reaches 0.

**Parameters:**

*surf* surface handle

**Returns:**

true on success

**Default action:**

None, returns false.

**Note:**

Derived classes should overload this function to decrement a surface reference counter and release the surface if required.

**See also:**

[clbkCreateSurface](#), [clbkIncrSurfaceRef](#)

**16.20.3.42 virtual bool oapi::GraphicsClient::clbkGetSurfaceSize (SURFHANDLE *surf*, DWORD \**w*, DWORD \**h*) [inline, virtual]**

Return the width and height of a surface.

**Parameters:**

← *surf* surface handle  
→ *w* surface width  
→ *h* surface height

**Returns:**

true if surface dimensions could be obtained.

**Default action:**

Sets w and h to 0 and returns false.

**See also:**

[clbkCreateSurface](#)

**16.20.3.43 virtual bool oapi::GraphicsClient::clbkSetSurfaceColourKey (SURFHANDLE *surf*, DWORD *ckey*) [inline, virtual]**

Set transparency colour key for a surface.

**Parameters:**

*surf* surface handle  
*ckey* transparency colour key value

**Default action:**

None, returns false.

**Note:**

Derived classes should overload this method if the renderer supports colour key transparency for surfaces.

**16.20.3.44 virtual DWORD oapi::GraphicsClient::clbkGetDeviceColour (BYTE *r*, BYTE *g*, BYTE *b*) [inline, virtual]**

Convert an RGB colour triplet into a device-specific colour value.

**Parameters:**

*r* red component  
*g* green component  
*b* blue component

**Returns:**

colour value

**Note:**

Derived classes should overload this method to convert RGB colour definitions into device-compatible colour values, taking into account the colour depth of the render device etc.

**Default action:**

Packs the RGB values into a DWORD of the form 0x00RRGGBB, with 8 bits per colour component.

**See also:**

[clbkFillSurface](#)

**16.20.3.45 virtual bool oapi::GraphicsClient::clbkBlt (SURFHANDLE *tgt*, DWORD *tgtx*, DWORD *tgy*, SURFHANDLE *src*, DWORD *flag* = 0) const [inline, virtual]**

Copy one surface into an area of another one.

**Parameters:**

*tgt* target surface handle  
*tgtx* left edge of target rectangle  
*tgy* top edge of target rectangle  
*src* source surface handle  
*flag* blitting parameters (see notes)

**Returns:**

true on success, false if the blit cannot be performed.

**Default action:**

None, returns false.

**Note:**

By convention, tgt==NULL is valid and refers to the primary render surface (e.g. for copying 2-D overlay surfaces).

The following bit-flags are defined:

BLT_SRCCOLORKEY	Use the colour key defined by the source surface for transparency
BLT_TGTCOLORKEY	Use the colour key defined by the target surface for transparency

If a client doesn't support some of the flags, it should quietly ignore it.

**See also:**

`clbkBlt(SURFHANDLE,DWORD,DWORD,SURFHANDLE,DWORD,DWORD,DWORD,DWORD,DWORD)`

**16.20.3.46 virtual bool oapi::GraphicsClient::clbkBlt (SURFHANDLE *tgt*, DWORD *tgtx*, DWORD *tgyt*, SURFHANDLE *src*, DWORD *srcx*, DWORD *srcy*, DWORD *w*, DWORD *h*, DWORD *flag* = 0) const [inline, virtual]**

Copy a rectangle from one surface to another.

**Parameters:**

*tgt* target surface handle  
*tgtx* left edge of target rectangle  
*tgyt* top edge of target rectangle  
*src* source surface handle  
*srcx* left edge of source rectangle  
*srcy* top edge of source rectangle  
*w* width of rectangle  
*h* height of rectangle  
*flag* blitting parameters (see notes)

**Returns:**

true on success, false if the blit cannot be performed.

**Default action:**

None, returns false.

**Note:**

By convention, *tgt*=NULL is valid and refers to the primary render surface (e.g. for copying 2-D overlay surfaces).

The following bit-flags are defined:

BLT_SRCCOLORKEY	Use the colour key defined by the source surface for transparency
BLT_TGTCOLORKEY	Use the colour key defined by the target surface for transparency

If a client doesn't support some of the flags, it should quietly ignore it.

**See also:**

`clbkBlt(SURFHANDLE,DWORD,DWORD,SURFHANDLE,DWORD)`

**16.20.3.47 virtual bool oapi::GraphicsClient::clbkScaleBlt (SURFHANDLE *tgt*, DWORD *tgtx*, DWORD *tgyt*, DWORD *tgtw*, DWORD *tght*, SURFHANDLE *src*, DWORD *srcx*, DWORD *srcy*, DWORD *srcw*, DWORD *srch*, DWORD *flag* = 0) const [inline, virtual]**

Copy a rectangle from one surface to another, stretching or shrinking as required.

**Parameters:**

*tgt* target surface handle

*tgtx* left edge of target rectangle  
*tgyt* top edge of target rectangle  
*tgtw* width of target rectangle  
*tght* height of target rectangle  
*src* source surface handle  
*srcx* left edge of source rectangle  
*srcy* top edge of source rectangle  
*srcw* width of source rectangle  
*srch* height of source rectangle  
*flag* blitting parameters

**Returns:**

true on success, false if the blit cannot be performed.

**Default action:**

None, returns false.

**Note:**

By convention, tgt==NULL is valid and refers to the primary render surface (e.g. for copying 2-D overlay surfaces).

**See also:**

[clbkBlt\(SURFHANDLE,DWORD,DWORD,SURFHANDLE,DWORD\)](#),  
[clbkBlt\(SURFHANDLE,DWORD,DWORD,SURFHANDLE,DWORD,DWORD,DWORD,DWORD,DWORD\)](#)

**16.20.3.48 virtual int oapi::GraphicsClient::clbkBeginBltGroup (SURFHANDLE tgt) [virtual]**

Begins a block of blitting operations to the same target surface.

**Parameters:**

*tgt* Target surface for subsequent blitting calls.

**Returns:**

Should return an error code (0 on success, the return value from the base class call, or a client-specific code)

**Default action:**

If no target is currently set, stores tgt in surfBltTgt and returns 0. Returns -2 if a target was already set. Returns -3 if tgt==RENDERTGT\_NONE

**Note:**

All blitting calls following this function use the same target until [clbkEndBltGroup](#) is called. Clients which have to perform initialisations for a new blitting target can do this here, and then assume for the following oapiBlit calls that the target is already initialised.

The special target RENDERTGT\_MAINWINDOW refers to the main render surface. Within the blitting block, multiple source surfaces may be used. No particular order or grouping according to source surface can be guaranteed. Blitting calls using a different target within a blitting group are not filtered out. It is up to the client how to handle this (honour the call, ignore it or throw an error)

**See also:**

[clbkEndBltGroup](#)

**16.20.3.49 virtual int oapi::GraphicsClient::clbkEndBltGroup () [virtual]**

Ends a block of blitting operations to the same target surface.

**Returns:**

Should return an error code (0 on success, the return value from the base class call, or a client-specific code)

**Default action:**

If surfBltTgt was set, clears it (to RENDERTGT\_NONE) and returns 0. Otherwise returns -2;

**See also:**

[clbkBeginBltGroup](#)

**16.20.3.50 virtual bool oapi::GraphicsClient::clbkFillSurface (SURFHANDLE *surf*, DWORD *col*) const [inline, virtual]**

Fill a surface with a uniform colour.

**Parameters:**

*surf* surface handle

*col* colour value

**Returns:**

true on success, false if the fill operation cannot be performed.

**Default action:**

None, returns false.

**Note:**

Parameter col is a device-dependent colour value (see [clbkGetDeviceColour](#)).

**See also:**

[clbkFillSurface\(SURFHANDLE,DWORD,DWORD,DWORD,DWORD,DWORD\)](#)

**16.20.3.51 virtual bool oapi::GraphicsClient::clbkFillSurface (SURFHANDLE *surf*, DWORD *tgx*, DWORD *tgy*, DWORD *w*, DWORD *h*, DWORD *col*) const [inline, virtual]**

Fill an area in a surface with a uniform colour.

**Parameters:**

*surf* surface handle

*tgx* left edge of target rectangle

*tgy* top edge of target rectangle

*w* width of rectangle

*h* height of rectangle

*col* colour value

**Returns:**

true on success, false if the fill operation cannot be performed.

**Default action:**

None, returns false.

**Note:**

Parameter col is a device-dependent colour value (see [clbkGetDeviceColour](#)).

**See also:**

[clbkFillSurface\(SURFHANDLE,DWORD\)](#)

**16.20.3.52 virtual bool oapi::GraphicsClient::clbkCopyBitmap (SURFHANDLE *pdds*, HBITMAP *hbm*, int *x*, int *y*, int *dx*, int *dy*) [virtual]**

Copy a bitmap object into a surface.

**Parameters:**

*pdds* surface handle

*hbm* bitmap handle

*x* left edge of source bitmap area to be copied

*y* top edge of source bitmap area to be copied

*dx* width of source bitmap area to be copied

*dy* height of source bitmap area to be copied

**Returns:**

*true* on success, *false* if surface or bitmap handle are invalid.

**Note:**

The source bitmap area is stretched as required to fit the area of the target surface.

**16.20.3.53 virtual Sketchpad\* oapi::GraphicsClient::clbkGetSketchpad (SURFHANDLE *surf*)** [inline, virtual]

Create a 2-D drawing object ("sketchpad") associated with a surface.

**Parameters:**

*surf* surface handle

**Returns:**

Pointer to drawing object.

**Default action:**

None, returns NULL.

**Note:**

Clients should overload this function to provide 2-D drawing support. This requires an implementation of a class derived from [Sketchpad](#) which provides the drawing context and drawing primitives.

**See also:**

[Sketchpad](#), [clbkReleaseSketchpad](#)

**16.20.3.54 virtual void oapi::GraphicsClient::clbkReleaseSketchpad (Sketchpad \**sp*)** [inline, virtual]

Release a drawing object.

**Parameters:**

*sp* pointer to drawing object

**Default action:**

None.

**See also:**

[Sketchpad](#), [clbkGetSketchpad](#)

**16.20.3.55 virtual Font\* oapi::GraphicsClient::clbkCreateFont (int *height*, bool *prop*, const char \**face*, oapi::Font::Style *style* = oapi::Font::NORMAL, int *orientation* = 0) const** [inline, virtual]

Create a font resource for 2-D drawing.

**Parameters:**

*height* cell or character height [pixel]

*prop* proportional/fixed width flag

*face* font face name

*style* font decoration style

*orientation* text orientation [1/10 deg]

**Returns:**

Pointer to font resource

**Default action:**

None, returns NULL.

**Note:**

For a description of the parameters, see [Font constructor oapi::Font::Font](#)

**See also:**

[clbkReleaseFont](#), [oapi::Font](#)

**16.20.3.56 virtual void oapi::GraphicsClient::clbkReleaseFont (Font \* *font*) const [inline, virtual]**

De-allocate a font resource.

**Parameters:**

*font* pointer to font resource

**Default action:**

None.

**See also:**

[clbkCreateFont](#), [oapi::Font](#)

**16.20.3.57 virtual Pen\* oapi::GraphicsClient::clbkCreatePen (int *style*, int *width*, DWORD *col*) const [inline, virtual]**

Create a pen resource for 2-D drawing.

**Parameters:**

*style* line style (0=invisible, 1=solid, 2=dashed)

*width* line width [pixel]

*col* line colour (format: 0xBBGGRR)

**Returns:**

Pointer to pen resource

**Default action:**

None, returns NULL.

**See also:**

[clbkReleasePen](#), [oapi::Pen](#)

**16.20.3.58 virtual void oapi::GraphicsClient::clbkReleasePen (Pen \* *pen*) const [inline, virtual]**

De-allocate a pen resource.

**Parameters:**

*pen* pointer to pen resource

**Default action:**

None.

**See also:**

[clbkCreatePen](#), [oapi::Pen](#)

**16.20.3.59 virtual Brush\* oapi::GraphicsClient::clbkCreateBrush (DWORD *col*) const [inline, virtual]**

Create a brush resource for 2-D drawing.

**Parameters:**

*col* line colour (format: 0xBBGGRR)

**Returns:**

Pointer to brush resource

**Default action:**

None, returns NULL.

**See also:**

[clbkReleaseBrush](#), [oapi::Brush](#)

**16.20.3.60 virtual void oapi::GraphicsClient::clbkReleaseBrush (Brush \* *brush*) const [inline, virtual]**

De-allocate a brush resource.

**Parameters:**

*brush* pointer to brush resource

**Default action:**

None.

**See also:**

[clbkCreateBrush](#), [oapi::Brush](#)

**16.20.3.61 virtual HDC oapi::GraphicsClient::clbkGetSurfaceDC (SURFHANDLE *surf*) [inline, virtual]**

Return a Windows graphics device interface handle for a surface.

**Parameters:**

*surf* surface handle

**Returns:**

GDI handle, or NULL on failure

**Default action:**

None, returns NULL.

**Note:**

Clients which can obtain a Windows GDI handle for a surface should overload this method.

**Todo**

This method should be moved into the GDIClient class

**16.20.3.62 virtual void oapi::GraphicsClient::clbkReleaseSurfaceDC (SURFHANDLE *surf*, HDC *hDC*) [inline, virtual]**

Release a Windows graphics device interface.

**Parameters:**

*surf* surface handle

*hDC* GDI handle

**Default action:**

None.

**Note:**

Clients which can obtain a Windows GDI handle for a surface should overload this method to release an existing GDI.

**Todo**

This method should be moved into the GDIClient class

**16.20.3.63 virtual bool oapi::GraphicsClient::clbkUseLaunchpadVideoTab () const [inline, protected, virtual]**

Launchpad video tab indicator.

Indicate if the the default video tab in the Orbiter launchpad dialog is to be used for obtaining user video preferences. If a derived class returns false here, the video tab is not shown.

**Returns:**

true if the module wants to use the video tab in the launchpad dialog, false otherwise.

**Default action:**

Return true.

**16.20.3.64 virtual HWND oapi::GraphicsClient::clbkCreateRenderWindow () [protected, virtual]**

Simulation session start notification.

Called at the beginning of a simulation session to allow the client to create the 3-D rendering window (or to switch into fullscreen mode).

**Returns:**

Should return window handle of the rendering window.

**Default action:**

For windowed mode, opens a window of the size specified by the VideoData structure (for fullscreen mode, opens a small dummy window) and returns the window handle.

**Note:**

For windowed modes, the viewW and viewH parameters should return the window client area size. For fullscreen mode, they should contain the screen resolution.

Derived classes should perform any required per-session initialisation of the 3D render environment here.

**16.20.3.65 virtual void oapi::GraphicsClient::clbkPostCreation () [inline, protected, virtual]**

Simulation startup finalisation.

Called at the beginning of a simulation session after the scenario has been parsed and the logical object have been created.

**Default action:**

None

**16.20.3.66 virtual void oapi::GraphicsClient::clbkCloseSession (bool *fastclose*) [inline, protected, virtual]**

End of simulation session notification.

Called before the end of a simulation session. At the point of call, logical objects still exist (OBJHANDLES valid), and external modules are still loaded.

**Parameters:**

*fastclose* Indicates a "fast shutdown" request (see notes)

**Default action:**

None.

**Note:**

Derived clients can use this function to perform cleanup operations for which the simulation objects are still required.

If fastclose == true, the user has selected one of the fast shutdown options (terminate Orbiter, or respawn Orbiter process). In this case, the current process will terminate, and the graphics client can skip object cleanup and deallocation in order to speed up the closedown process.

**See also:**

[clbkDestroyRenderWindow](#)

**16.20.3.67 virtual void oapi::GraphicsClient::clbkDestroyRenderWindow (bool *fastclose*) [protected, virtual]**

Render window closure notification.

Called at the end of a simulation session to allow the client to close the 3-D rendering window (or to switch out of fullscreen mode) and clean up the session environment. At the point of call, all logical simulation objects have been destroyed, and object modules have been unloaded. This method should not access any OBJHANDLE or [VESSEL](#) objects any more. For closedown operations that require access to the simulation objects, use clbkCloseSession instead.

**Parameters:**

*fastclose* Indicates a "fast shutdown" request (see notes)

**Default action:**

None.

**Note:**

Derived classes should perform any required cleanup of the 3D render environment here.

The user may change the video parameters before starting a new simulation session. Therefore, device-specific options should be destroyed and re-created at the start of the next session.

If fastclose == true, the user has selected one of the fast shutdown options (terminate Orbiter, or respawn Orbiter process). In this case, the current process will terminate, and the graphics client can skip object cleanup and deallocation in order to speed up the closedown process.

**See also:**

[clbkCloseSession](#)

**16.20.3.68 virtual void oapi::GraphicsClient::clbkUpdate (bool *running*) [inline, protected, virtual]**

Per-frame update notification.

Called once per frame, after the logical world state has been updated, but before [clbkRenderScene\(\)](#), to allow the client to perform any logical state updates.

**Parameters:**

*running* true if simulation is running, false if paused.

**Default action:**

None.

**Note:**

Unlike clbkPreStep and clbkPostStep, this method is also called while the simulation is paused.

**16.20.3.69 virtual void oapi::GraphicsClient::clbkRenderScene () [protected, pure virtual]**

Per-frame render notification.

Called once per frame, after the logical world state has been updated, to allow the client to render the current scene.

**Note:**

This method is also called continuously while the simulation is paused, to allow camera panning (although in that case the logical world state won't change between frames).

After the 3D scene has been rendered, this function should call [Render2DOverlay](#) to initiate rendering of 2D elements (2D instrument panel, HUD, etc.)

**16.20.3.70 virtual bool oapi::GraphicsClient::clbkDisplayFrame () [inline, protected, virtual]**

Display a scene on screen after rendering it.

Called after clbkRenderScene to allow the client to display the rendered scene (e.g. by page-flipping, or blitting from background to primary frame buffer. This method can also be used by the client to display any top-level 2-D overlays (e.g. dialogs) on the primary frame buffer.

**Returns:**

Should return true on successful operation, false on failure or if no operation was performed.

**Default action:**

None, returns false.

**16.20.3.71 void oapi::GraphicsClient::Render2DOverlay () [protected]**

Notifies Orbiter to to initiate rendering of the 2D scene overlay.

The 2D overlay is used to render 2D instrument panels, HUD, the info boxes at the top left and right of the screen, etc. This function should typically be called at the end of [clbkRenderScene](#), after the 3D scene has been rendered, but before the rendering environment is released. During the execution of this function, Orbiter will call the [clbkRender2DPanel](#) function several times to allow the client to build up the 2D layer.

**Note:**

Orbiter will *not* acquire a [Sketchpad](#) environment while executing this function, because the graphics driver may not allow to lock surfaces for drawing while in render mode. If a [Sketchpad](#) environment is required to draw on top of the render window (for example for displaying specific HUD elements), it is acquired after clbkRenderScene returns.

**See also:**

[clbkRenderScene](#), [clbkRender2DPanel](#)

**16.20.3.72 virtual void oapi::GraphicsClient::clbkStoreMeshPersistent (MESHHANDLE *hMesh*, const char \**fname*) [inline, protected, virtual]**

Store a persistent mesh template.

Called when a plugin loads a mesh with [oapiLoadMeshGlobal](#), to allow the client to store a copy of the mesh in client-specific format. Whenever the mesh is required later, the client can create an instance as a copy of the template, rather than creating it by converting from Orbiter's mesh format.

**Parameters:**

*hMesh* mesh handle

*fname* mesh file name

**Default action:**

None.

**Note:**

Use [oapiMeshGroup](#) to obtain mesh data and convert them to a suitable format.

the mesh templates loaded with [oapiLoadMeshGlobal](#) are shared between all vessel instances and should never be edited. Vessels should make individual copies of the mesh before modifying them (e.g. for animations)

The file name is provided to allow the client to parse the mesh directly from file, rather than copying it from the hMesh object, or to use an alternative mesh file.

The file name contains a path relative to Orbiter's main mesh directory.

**16.20.3.73 HWND oapi::GraphicsClient::LaunchpadVideoTab () const [inline, protected]**

Returns the window handle of the 'video' tab of the Orbiter Launchpad dialog.

If [clbkUseLaunchpadVideoTab\(\)](#) is overloaded to return false, this function will return NULL.

**16.20.3.74 DWORD oapi::GraphicsClient::LoadStars (DWORD *n*, StarRec \**rec*)**

Load star data from Orbiter's data base file.

Load up to '*n*' data records from the default data base (in decreasing order of apparent magnitude).

**Parameters:**

*n* Requested number of stars

*rec* Pointer to an array receiving the data.

**Returns:**

The actual number of loaded stars

**Note:**

*rec* must be allocated to size  $\geq n$  on call.

**16.20.3.75 DWORD oapi::GraphicsClient::LoadConstellationLines (DWORD *n*, ConstRec \* *rec*)**

Load constellation line data from Orbiter's data base file.

Load up to '*n*' constellation lines from the default constellation data base.

**Parameters:**

*n* Requested number of lines

*rec* Pointer to an array receiving the data.

**Returns:**

The actual number of lines loaded.

**Note:**

*rec* must be allocated to size  $\geq n$  on call.

**16.20.3.76 DWORD oapi::GraphicsClient::GetCelestialMarkers (const LABELLIST \*\* *cm\_list*) const**

Returns an array of celestial marker lists.

**Parameters:**

*cm\_list* array of marker lists

**Returns:**

number of lists in the array

**See also:**

[LABELLIST](#)

**16.20.3.77 DWORD oapi::GraphicsClient::GetSurfaceMarkers (OBJHANDLE *hObj*, const LABELLIST \*\* *sm\_list*) const**

Returns an array of surface marker lists for a planet.

**Parameters:**

*hObj* planet handle

*sm\_list* array of marker lists

**Returns:**

number of lists in the array

**Note:**

*hObj* must refer to a planet or moon. Other objects are not supported.

**See also:**

[LABELLIST](#)

The documentation for this class was generated from the following file:

- Orbitersdk/include/GraphicsAPI.h

## 16.21 oapi::GraphicsClient::LABELLIST Struct Reference

```
#include <GraphicsAPI.h>
```

### 16.21.1 Detailed Description

Label list description for celestial and surface markers.

#### Public Attributes

- char **name** [64]  
*list name*
- LABELSPEC \* **list**  
*marker array*
- int **length**  
*length of the marker array*
- int **colour**  
*marker colour index (0-5)*
- int **shape**  
*marker shape index (0-4)*
- float **size**  
*marker size factor*
- float **distfac**  
*marker distance cutout factor*
- DWORD **flag**  
*reserved*
- bool **active**  
*active list flag*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/GraphicsAPI.h

## 16.22 oapi::GraphicsClient::VIDEODATA Struct Reference

```
#include <GraphicsAPI.h>
```

### 16.22.1 Detailed Description

Structure containing default video options, as stored in Orbiter.cfg.

#### Public Attributes

- bool **fullscreen**  
*fullscreen mode flag*
- bool **forceenum**  
*enforce device enumeration flag*
- bool **try stencil**  
*stencil buffer flag*
- bool **novsync**  
*no vsync flag*
- bool **pageflip**  
*allow page flipping in fullscreen*
- int **deviceidx**  
*video device index*
- int **modeidx**  
*video mode index*
- int **winw**  
*window width*
- int **winh**  
*window height*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/GraphicsAPI.h

## 16.23 GraphMFD Class Reference

```
#include <MFDAPI.h>
```

Inheritance diagram for GraphMFD:

Collaboration diagram for GraphMFD:

### 16.23.1 Detailed Description

This class is derived from [MFD](#) and provides a template for [MFD](#) modes containing 2D graphs.

This class is derived from [MFD](#) and provides a template for [MFD](#) modes containing 2D graphs. An example is the ascent profile recorder in the samples\CustomMFD folder.

#### Public Member Functions

- **GraphMFD** (DWORD w, DWORD h, [VESSEL](#) \*vessel)  
*Constructor. Creates a new [GraphMFD](#).*
- int **AddGraph** (void)  
*Adds a new graph to the [MFD](#).*
- void **AddPlot** (int g, float \*absc, float \*data, int ndata, int col, int \*ofs=0)  
*Adds a plot to an existing graph.*
- void **SetRange** (int g, int axis, float rmin, float rmax)  
*Sets a fixed range for the x or y axis of a graph.*
- void **SetAutoRange** (int g, int axis, int p=-1)  
*Allows the graph to set its range automatically according to the range of the plots.*
- void **SetAutoTicks** (int g, int axis)  
*Calculates tick intervals for a given graph and axis.*
- void **SetAxisTitle** (int g, int axis, char \*title)  
*Sets the title for a given graph and axis.*
- void **Plot** (HDC hDC, int g, int h0, int h1, const char \*title=0)  
*Displays a graph.*
- void **FindRange** (float \*d, int ndata, float &dmin, float &dmax) const  
*Determines the range of an array of data.*

### Protected Attributes

- int **ngraph**
- struct GraphMFD::GRAPH \* **graph**

#### 16.23.2 Constructor & Destructor Documentation

##### 16.23.2.1 GraphMFD::GraphMFD (DWORD *w*, DWORD *h*, VESSEL \* *vessel*)

Constructor. Creates a new [GraphMFD](#).

###### Parameters:

- w* width of the [MFD](#) display (pixel)
- h* height of the [MFD](#) display (pixel)
- vessel* pointer to [VESSEL](#) interface associated with the [MFD](#)

#### 16.23.3 Member Function Documentation

##### 16.23.3.1 int GraphMFD::AddGraph (void)

Adds a new graph to the [MFD](#).

###### Returns:

graph identifier

###### Note:

This function allocates data for a new graph. To display plots in the new graph, one or more calls to [AddPlot](#) are required.

##### 16.23.3.2 void GraphMFD::AddPlot (int *g*, float \* *absc*, float \* *data*, int *ndata*, int *col*, int \* *ofs* = 0)

Adds a plot to an existing graph.

###### Parameters:

- g* graph identifier
- absc* pointer to array containing the abscissa (x-axis) values.
- data* pointer to array containing the data (y-axis) values.
- ndata* number of data points
- col* plot colour index
- ofs* pointer to data offset (optional)

###### Note:

Data arrays are not copied, so they should not be deleted after the call to [AddPlot\(\)](#).  
*col* is used as an index to select a pen for the plot using the [SelectDefaultPen](#) function. Valid range is the same as for [SelectDefaultPen\(\)](#).  
If defined, *ofs* is the index of the first plot value in the data array. The plot is drawn using the points *ofs* to *ndata*-1, followed by points 0 to *ofs*-1. This allows to define continuously updated plots without having to copy blocks of data within the arrays.

**16.23.3.3 void GraphMFD::SetRange (int *g*, int *axis*, float *rmin*, float *rmax*)**

Sets a fixed range for the x or y axis of a graph.

**Parameters:**

*g* graph identifier  
*axis* axis identifier (0=x, 1=y)  
*rmin* minimum value  
*rmax* maximum value

**Note:**

The range applies to all plots in the graph.

**16.23.3.4 void GraphMFD::SetAutoRange (int *g*, int *axis*, int *p* = -1)**

Allows the graph to set its range automatically according to the range of the plots.

**Parameters:**

*g* graph identifier  
*axis* axis identifier (0=x, 1=y)  
*p* plot identifier (-1=all)

**Note:**

If *p*>=0, then *p* specifies the plot used for determining the graph range. If *p* = -1, then all of the graph's plots are used to determine the range.

**16.23.3.5 void GraphMFD::SetAutoTicks (int *g*, int *axis*)**

Calculates tick intervals for a given graph and axis.

**Parameters:**

*g* graph identifier  
*axis* axis identifier (0=x, 1=y)

**Note:**

This function is called from within SetRange and normally doesn't need to be called explicitly by derived classes.

**16.23.3.6 void GraphMFD::SetTitle (int *g*, int *axis*, char \* *title*)**

Sets the title for a given graph and axis.

**Parameters:**

*g* graph identifier  
*axis* axis identifier (0=x, 1=y)

*title* axis title

**Note:**

The **MFD** may append an extension of the form "x <scale>" to the title, where <scale> is a scaling factor applied to the tick labels of the axis. It is therefore a good idea to finish the title with the units applicable to the data of this axis, so that for example a title "Altitude: km" may become "Altitude: km x 1000".

#### 16.23.3.7 void GraphMFD::Plot (HDC *hDC*, int *g*, int *h0*, int *h1*, const char \* *title* = 0)

Displays a graph.

**Parameters:**

*hDC* Windows device context

*g* graph identifier

*h0* upper boundary of plot area (pixel)

*h1* lower boundary of plot area (pixel)

*title* graph title

**Note:**

This function should be called from Update to paint the graph(s) into the provided device context.

#### 16.23.3.8 void GraphMFD::FindRange (float \* *d*, int *ndata*, float & *dmin*, float & *dmax*) const

Determines the range of an array of data.

**Parameters:**

*d* data array

*ndata* number of data

*dmin* minimum value on return

*dmax* maximum value on return

The documentation for this class was generated from the following file:

- Orbitersdk/include/**MFDAPIL.h**

## 16.24 GROUPEDITSPEC Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for GROUPEDITSPEC:

### 16.24.1 Detailed Description

Structure used by [oapiEditMeshGroup](#) to define the group elements to be replaced.

**Note:**

Only the group elements specified in the *flags* entry will be replaced or modified. The elements that are to remain unchanged can be left undefined in the [GROUPEDITSPEC](#) structure. For example, if only GRPEDIT\_VTXCRDX is specified, only the 'x' fields in the Vtx array need to be assigned. to replace individual vertices in the group, the nVtx entry should contain the number of vertices to be replaced, the vIdx array should contain the indices ( $\geq 0$ ) of the vertices to be replaced, and Vtx should contain the new vertex values of those vertices. If vIdx==NULL, vertices are replaced in sequence from the beginning of the group's vertex list. nVtx must be less or equal the number of vertices in the group.

**See also:**

[oapiEditMeshGroup](#), Mesh group editing flags

#### Public Attributes

- DWORD [flags](#)  
*flags (see [Mesh group editing flags](#))*
- DWORD [UsrFlag](#)  
*Replacement for group UsrFlag entry.*
- [NTVERTEX \\* Vtx](#)  
*Replacement for group vertices.*
- DWORD [nVtx](#)  
*Number of vertices to be replaced.*
- WORD \* [vIdx](#)  
*Index list for vertices to be replaced.*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 16.25 HELPCONTEXT Struct Reference

```
#include <OrbiterAPI.h>
```

### 16.25.1 Detailed Description

Context information for an Orbiter ingame help page.

**See also:**

[oapiOpenHelp](#)

### Public Attributes

- char \* **helpfile**
- char \* **topic**
- char \* **toc**
- char \* **index**

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 16.26 HUDPARAM Union Reference

```
#include <OrbiterAPI.h>
```

### 16.26.1 Detailed Description

Mode-specific parameters for HUD mode settings.

**See also:**

[oapiSetHUDMode\(int,const HUDPARAM\\*\)](#)

### Public Attributes

- struct {  
    **OBJHANDLE hRef**  
        *orbit HUD reference object (NULL for auto)*  
    } **HUDorbit**
- struct {  
    **DWORD NavIdx**  
        *docking HUD nav receiver index (>= 0)*  
    } **HUDDocking**

The documentation for this union was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 16.27 oapi::IVECTOR2 Union Reference

```
#include <DrawAPI.h>
```

### 16.27.1 Detailed Description

Integer-valued 2-D vector type.

**Note:**

This structure is designed to be compatible with the Windows POINT type.

**Public Attributes**

- long [data](#) [2]  
*vector data array*
- struct {
   
   long [x](#)  
*vector x coordinate*
  
   long [y](#)  
*vector y coordinate*
  
 };

The documentation for this union was generated from the following file:

- OrbiterSDK/include/DrawAPI.h

## 16.28 LaunchpadItem Class Reference

```
#include <OrbiterAPI.h>
```

### 16.28.1 Detailed Description

Base class to define launchpad items.

[LaunchpadItem](#) is the base class for objects that can be inserted into the parameter list of the Extra tab of the Orbiter Launchpad dialog. The Extra tab provides a mechanism for plugin modules to allow users to set global parameters specific to an addon. [LaunchpadItem](#) is notified whenever the user selects the item from the list, and when parameters need to be read from or written to disk.

**See also:**

[oapiRegisterLaunchpadItem](#), [oapiUnregisterLaunchpadItem](#)

### Public Member Functions

- [LaunchpadItem \(\)](#)  
*Constructor: Creates a new launchpad item.*
- virtual [~LaunchpadItem \(\)](#)  
*Destructor: Destroys the launchpad item.*
- virtual char \* [Name \(\)](#)  
*Derived classes should return a pointer to the string to appear in the Launchpad "Extra" list.*
- virtual char \* [Description \(\)](#)  
*Derived classes should return a pointer to the string containing a description of the item. The description is shown next to the Launchpad list whenever the item is selected.*
- virtual bool [OpenDialog](#) (HINSTANCE hInst, HWND hLaunchpad, int resId, DLGPROC pDlg)  
*Opens a dialog box associated with the launchpad item.*

- virtual bool [clbkOpen](#) (HWND hLaunchpad)

*This method is called whenever the user opens the item by double-clicking on the list or clicking the "Edit" button below the list.*

- virtual int [clbkWriteConfig](#) ()

*This method is called whenever the item should write its current state to a file.*

## Public Attributes

- [LAUNCHPADITEM\\_HANDLE hItem](#)

### 16.28.2 Member Function Documentation

#### 16.28.2.1 virtual char\* LaunchpadItem::Name () [virtual]

Derived classes should return a pointer to the string to appear in the Launchpad "Extra" list.

##### Returns:

Pointer to the item label in the list.

**Default action:** Returns NULL (no entry in the list).

#### 16.28.2.2 virtual char\* LaunchpadItem::Description () [virtual]

Derived classes should return a pointer to the string containing a description of the item. The description is shown next to the Launchpad list whenever the item is selected.

##### Returns:

Pointer to the descriptive string, or NULL if there is none.

**Default action:** Returns NULL (no description).

##### Note:

Line breaks can be inserted into the description with a carriage return/newline sequence (\r\n).

#### 16.28.2.3 virtual bool LaunchpadItem::OpenDialog (HINSTANCE hInst, HWND hLaunchpad, int resId, DLGPROC pDlg) [virtual]

Opens a dialog box associated with the launchpad item.

##### Parameters:

*hInst* module instance handle

*hLaunchpad* launchpad window handle

*resId* integer resource ID of the dialog box

*pDlg* dialog box message handler

##### Returns:

Currently this function always returns *true*.

**Note:**

This function is usually called in the body of [LaunchpadItem::clbkOpen\(\)](#).

It is an alternative to the standard Windows DialogBox function. It has the advantage that a pointer to the [LaunchpadItem](#) instance is passed as IParam to the message handler with the WM\_INITDIALOG message. In all subsequent calls to the handler, the [LaunchpadItem](#) instance pointer can be obtained with a call to *GetWindowLong (hWnd, DWL\_USER)*, where hWnd is the dialog box handle passed to the message handler.

**16.28.2.4 virtual bool LaunchpadItem::clbkOpen (HWND *hLaunchpad*) [virtual]**

This method is called whenever the user opens the item by double-clicking on the list or clicking the "Edit" button below the list.

**Parameters:**

***hLaunchpad*** The window handle of the Launchpad dialog

**Returns:**

Currently ignored. Should be *true* if the derived class processes this callback function.

**Default action:** Nothing; returns false.

**Note:**

The derived class can use this function to open a dialog box or some other means of allowing the user to set addon-specific parameters.

**16.28.2.5 virtual int LaunchpadItem::clbkWriteConfig () [virtual]**

This method is called whenever the item should write its current state to a file.

**Returns:**

Currently ignored. Should be 0.

**Default action:** Nothing; returns 0.

**Note:**

This function is called before a simulation session is launched, before Orbiter shuts down, and before the module is deactivated. It allows the module to write its current state to a file, so it can re-load its settings the next time Orbiter is launched.

You can either use default C or C++ methods to open a file for output, or you can use the [oapiOpenFile\(\)](#) method.

Modules should never write to the global Orbiter.cfg configuration file. Any addons that are not active when Orbiter overwrites Orbiter.cfg will lose their settings, since their [clbkWriteConfig\(\)](#) method cannot be called.

The best place to read the settings stored during a previous session is in the overloaded [LaunchpadItem](#) constructor. Use [oapiOpenFile](#) or another file access method compatible with the way the file was written. The parameter settings should then be stored in class member variables, and modified by user interaction.

The documentation for this class was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 16.29 LightEmitter Class Reference

```
#include <OrbiterAPI.h>
```

Inheritance diagram for LightEmitter:

Collaboration diagram for LightEmitter:

### 16.29.1 Detailed Description

Base class for defining a light source that can illuminate other objects.

#### Public Types

- enum **TYPE** { **LT\_NONE**, **LT\_POINT**, **LT\_SPOT**, **LT\_DIRECTIONAL** }

#### Public Member Functions

- **LightEmitter ()**  
*Create a light source with default parameters.*
- **LightEmitter (COLOUR4 diffuse, COLOUR4 specular, COLOUR4 ambient)**  
*Create a light source with specific colour parameters.*
- **TYPE GetType () const**  
*Returns the light source type.*
  - const **COLOUR4 & GetDiffuseColour () const**
  - const **COLOUR4 & GetSpecularColour () const**
  - const **COLOUR4 & GetAmbientColour () const**
  - void **Activate (bool act)**

*Activate or deactivate the light source.*

- `bool IsActive () const`  
*Returns activation status of light source.*
- `void SetPosition (const VECTOR3 &p)`  
*Set light source position.*
- `VECTOR3 GetPosition () const`  
*Returns the current source position.*
- `void SetPositionRef (const VECTOR3 *p)`  
*Set the reference pointer to the light source position.*
- `const VECTOR3 *GetPositionRef () const`  
*Returns a pointer to the position reference variable.*
- `void ShiftExplicitPosition (const VECTOR3 &ofs)`  
*Adds an offset to the explicit position definition of the source.*
- `void SetDirection (const VECTOR3 &d)`  
*Set light source direction.*
- `VECTOR3 GetDirection () const`  
*Returns the current source direction.*
- `void SetDirectionRef (const VECTOR3 *d)`  
*Set the reference pointer to the light source direction.*
- `const VECTOR3 *GetDirectionRef () const`  
*Returns a pointer to the direction reference variable.*
- `void SetIntensity (double in)`
- `double GetIntensity () const`
- `void SetIntensityRef (double *pin)`
- `const double *GetIntensityRef () const`

### Protected Member Functions

- `OBJHANDLE Attach (OBJHANDLE hObj)`
- `OBJHANDLE Detach ()`

### Protected Attributes

- `TYPE ltype`
- `OBJHANDLE hRef`
- `bool active`
- `COLOUR4 col_diff`
- `COLOUR4 col_spec`
- `COLOUR4 col_ambi`

- const double \* **intens**
- double **lintens**
- const **VECTOR3** \* **pos**
- const **VECTOR3** \* **dir**
- **VECTOR3** **lpos**
- **VECTOR3** **ldir**

### 16.29.2 Constructor & Destructor Documentation

#### 16.29.2.1 LightEmitter::LightEmitter ()

Create a light source with default parameters.

**Note:**

Creates a light source with white spectrum for diffuse, specular and emissive colour components. Intensity is set to 1, position (for point source objects) is set to (0,0,0) and direction (for spot and directional lights) is set to (0,0,1). To change these, use [SetPosition](#), [SetPositionRef](#), [SetDirection](#), [SetDirectionRef](#), [SetIntensity](#), [SetIntensityRef](#)

#### 16.29.2.2 LightEmitter::LightEmitter (COLOUR4 *diffuse*, COLOUR4 *specular*, COLOUR4 *ambient*)

Create a light source with specific colour parameters.

**Parameters:**

*diffuse* light source's contribution to lit objects' diffuse colour component

*specular* light source's contribution to lit objects' specular colour component

*ambient* light source's contribution to lit objects' ambient colour component

**Note:**

Intensity is set to 1, position (for point source objects) is set to (0,0,0) and direction (for spot and directional lights) is set to (0,0,1). To change these, use [SetPosition](#), [SetPositionRef](#), [SetIntensity](#), [SetIntensityRef](#)

### 16.29.3 Member Function Documentation

#### 16.29.3.1 void LightEmitter::Activate (bool *act*)

Activate or deactivate the light source.

**Parameters:**

*act* if *true*, activates the light source. Otherwise, deactivates the light source

**See also:**

[IsActive](#)

**16.29.3.2 bool LightEmitter::IsActive () const**

Returns activation status of light source.

**Returns:**

*true* if source is active, *false* otherwise.

**See also:**

[Activate](#)

**16.29.3.3 void LightEmitter::SetPosition (const VECTOR3 & p)**

Set light source position.

**Parameters:**

*p* new position [**m**] (in object or global coordinates)

**Note:**

The source position is only relevant for point and spot lights. It is ignored for directional lights. If the source is attached to an object (see [Attach](#)) the position is interpreted in the local object coordinates. Otherwise, the position is taken to be in global coordinates. After a displacement of the vessel's centre of mass (see [VESSEL::ShiftCG](#)), all light sources that define their position explicitly (via [SetPosition](#)) are updated automatically. Light sources with implicit position definition (via [SetPositionRef](#)) must update their positions themselves.

**See also:**

[GetPosition](#), [SetPositionRef](#), [GetPositionRef](#)

**16.29.3.4 VECTOR3 LightEmitter::GetPosition () const [inline]**

Returns the current source position.

**Returns:**

Current source position [**m**]

**Note:**

The source position is only relevant for point and spot lights. It is ignored for directional lights. If the source is attached to an object (see [Attach](#)) the returned vector is the source position in local object coordinates. Otherwise, the returned vector is the global source position.

**See also:**

[GetPosition](#), [SetPositionRef](#), [GetPositionRef](#)

**16.29.3.5 void LightEmitter::SetPositionRef (const VECTOR3 \**p*)**

Set the reference pointer to the light source position.

**Parameters:**

*p* pointer to vector defining the source position

**Note:**

This method links the position of the light source to an externally defined vector. By modifying the vector elements, the light source can be re-positioned instantly.

The vector variable pointed to by *p* must remain valid for the lifetime of the light source.

The source position is only relevant for point and spot lights. It is ignored for directional lights

**See also:**

[SetPosition](#), [GetPosition](#), [GetPositionRef](#)

**16.29.3.6 const VECTOR3\* LightEmitter::GetPositionRef () const**

Returns a pointer to the position reference variable.

**Returns:**

Pointer to the variable defining the light source position

**Note:**

If the position is defined explicitly (see [SetPosition](#)), this method simply returns a pointer to the lpos member variable. Otherwise, it returns the pointer specified in [SetPositionRef](#).

**See also:**

[SetPosition](#), [SetPositionRef](#), [GetPosition](#)

**16.29.3.7 void LightEmitter::ShiftExplicitPosition (const VECTOR3 & *ofs*)**

Adds an offset to the explicit position definition of the source.

**Parameters:**

*ofs* offset vector in local vessel coordinates

**Note:**

This method has only an effect for light sources whose positions are defined explicitly (via [SetPosition](#)). If the source position is defined implicitly (via [SetPositionRef](#)), this method has no effect. Modules that define their light source positions via implicit references must keep the positions up to date themselves (e.g. reacting to shifts in the centre of gravity).

**See also:**

[SetPosition](#), [SetPositionRef](#), [VESSEL::ShiftCG](#)

**16.29.3.8 void LightEmitter::SetDirection (const VECTOR3 & *d*)**

Set light source direction.

**Parameters:**

*p* new direction (in object or global coordinates)

**Note:**

The vector argument should be normalised to length 1.

The source direction is only relevant for spot and directional lights. It is ignored for point lights.

If the source is attached to an object (see Attach) the direction is interpreted in the local object coordinates. Otherwise, the direction is taken to be in global coordinates.

**See also:**

[GetDirection](#), [SetDirectionRef](#), [GetDirectionRef](#)

**16.29.3.9 VECTOR3 LightEmitter::GetDirection () const**

Returns the current source direction.

**Returns:**

Current source direction.

**Note:**

The source direction is only relevant for spot and directional lights. It is ignored for point lights.

If the source is attached to an object (see Attach) the returned vector is the source direction in local object coordinates. Otherwise, the returned vector is the global source direction.

**See also:**

[SetDirection](#), [SetDirectionRef](#), [GetDirectionRef](#)

**16.29.3.10 void LightEmitter::SetDirectionRef (const VECTOR3 \* *d*)**

Set the reference pointer to the light source direction.

**Parameters:**

*d* pointer to vector defining the source direction

**Note:**

This method links the direction of the light source to an externally defined vector. By modifying the vector elements, the light source can be re-directed instantly.

The vector variable pointed to by *d* must remain valid for the lifetime of the light source.

The source direction is only relevant for spot and directional lights. It is ignored for point lights

**See also:**

[SetDirection](#), [GetDirection](#), [GetDirectionRef](#)

**16.29.3.11 const VECTOR3\* LightEmitter::GetDirectionRef () const**

Returns a pointer to the direction reference variable.

**Returns:**

Pointer to the variable defining the light source direction

**Note:**

If the direction is defined explicitly (see [SetDirection](#)), this method simply returns a pointer to the ldir member variable. Otherwise, it returns the pointer specified in [SetDirectionRef](#).

**See also:**

[SetDirection](#), [SetDirectionRef](#), [GetDirection](#)

The documentation for this class was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

**16.30 LISTENTRY Struct Reference**

```
#include <OrbiterAPI.h>
```

**16.30.1 Detailed Description**

Entry specification for selection list entry.

**Public Attributes**

- char **name** [64]  
*entry string*
- DWORD **flag**  
*entry flags*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

**16.31 MATERIAL Struct Reference**

```
#include <OrbiterAPI.h>
```

Collaboration diagram for MATERIAL:

### 16.31.1 Detailed Description

material definition

#### Public Attributes

- [COLOUR4 diffuse](#)  
*diffuse component*
- [COLOUR4 ambient](#)  
*ambient component*
- [COLOUR4 specular](#)  
*specular component*
- [COLOUR4 emissive](#)  
*emissive component*
- float [power](#)  
*specular power*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 16.32 MATRIX3 Union Reference

```
#include <OrbiterAPI.h>
```

### 16.32.1 Detailed Description

3x3-element matrix

### Public Attributes

- double **data** [9]  
*array data interface (row-sorted)*
- struct {  
    double **m11**  
    double **m12**  
    double **m13**  
    double **m21**  
    double **m22**  
    double **m23**  
    double **m31**  
    double **m32**  
    double **m33**  
};  
  
*named data interface*

The documentation for this union was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 16.33 MESHGROUP Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for MESHGROUP:

### 16.33.1 Detailed Description

Defines a mesh group (subset of a mesh).

A mesh group contains a vertex list, an index list, a material and texture index, and a set of flags.

### Public Attributes

- **NTVERTEX** \* **Vtx**  
*vertex list*
- **WORD** \* **Idx**  
*index list*

- DWORD [nVtx](#)  
*vertex count*
- DWORD [nIdx](#)  
*index count*
- DWORD [MtrlIdx](#)  
*material index (>= 1, 0=none)*
- DWORD [TexIdx](#)  
*texture index (>= 1, 0=none)*
- DWORD [UsrFlag](#)  
*user-defined flag*
- WORD [zBias](#)  
*z bias*
- WORD [Flags](#)  
*internal flags*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 16.34 MESHGROUP\_TRANSFORM Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for MESHGROUP\_TRANSFORM:

### 16.34.1 Detailed Description

This structure defines an affine mesh group transform (translation, rotation or scaling).

See also:

[VESSEL::MeshgroupTransform](#)

### Public Types

- enum { **TRANSLATE**, **ROTATE**, **SCALE** }

### Public Attributes

- union {
   
    struct {
   
        VECTOR3 ref
   
            *rotation reference point*
  
        VECTOR3 axis
   
            *rotation axis direction*
  
        float angle
   
            *rotation angle [rad]*
  
    } rotparam
   
    struct {
   
        VECTOR3 shift
   
            *translation vector*
  
    } transparam
   
    struct {
   
        VECTOR3 scale
   
            *scaling factor*
  
    } scaleparam
   
}
- int nmesh
   
    *mesh index (>= 0)*
- int ngrp
   
    *group index (>= 0, or < 0 to indicate entire mesh)*
- enum MESHGROUP\_TRANSFORM:: { ... } transform
   
    *transformation flag*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/OrbiterAPI.h

## 16.35 MESHGROUPEX Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for MESHGROUPEX:

### 16.35.1 Detailed Description

extended mesh group definition

### Public Attributes

- **NTVERTEX \* Vtx**  
*vertex list*
- **WORD \* Idx**  
*index list*
- **DWORD nVtx**  
*vertex count*
- **DWORD nIdx**  
*index count*
- **DWORD MtrlIdx**  
*material index (>= 1, 0=none)*
- **DWORD TexIdx**  
*texture index (>= 1, 0=none)*
- **DWORD UsrFlag**  
*user-defined flag*
- **WORD zBias**  
*z bias*
- **WORD Flags**  
*internal flags*
- **DWORD TexIdxEx [MAXTEX]**  
*additional texture indices*
- **float TexMixEx [MAXTEX]**  
*texture mix values*

The documentation for this struct was generated from the following file:

- OrbiterSDK/include/OrbiterAPI.h

### 16.36 MFD Class Reference

```
#include <MFDAPI.h>
```

Inheritance diagram for MFD:

Collaboration diagram for MFD:

### 16.36.1 Detailed Description

This class acts as an interface for user defined MFD (multi functional display) modes.

This class acts as an interface for user defined MFD (multi functional display) modes. It provides control over keyboard and mouse functions to manipulate the MFD mode, and allows the module to draw the MFD display. The MFD class is a pure virtual class. Each userdefined MFD mode requires the definition of a specialised class derived from MFD. An example for a generic MFD mode implemented as a plugin module can be found in orbitersdk\samples\CustomMFD.

#### Public Member Functions

- **MFD** (DWORD w, DWORD h, [VESSEL](#) \*vessel)  
*Constructor. Creates a new [MFD](#).*
- virtual [~MFD](#) ()  
*[MFD](#) destructor.*
- virtual void [Update](#) (HDC hDC)=0  
*Callback function: Orbiter calls this method when the [MFD](#) needs to update its display.*
- void [InvalidateDisplay](#) ()  
*Force a display update in the next frame.*
- void [InvalidateButtons](#) ()  
*Force the [MFD](#) buttons to be redrawn.*
- void [Title](#) (HDC hDC, const char \*title) const  
*Displays a title string in the upper left corner of the [MFD](#) display.*
- HPEN [SelectDefaultPen](#) (HDC hDC, DWORD i) const  
*Selects a predefined pen into the device context.*
- HFONT [SelectDefaultFont](#) (HDC hDC, DWORD i) const  
*Selects a predefined [MFD](#) font into the device context.*
- virtual bool [ConsumeKeyBuffered](#) (DWORD key)  
*[MFD](#) keyboard handler for buffered keys.*
- virtual bool [ConsumeKeyImmediate](#) (char \*kstate)

*MFD keyboard handler for immediate (unbuffered) keys.*

- virtual bool [ConsumeButton](#) (int bt, int event)  
*MFD button handler.*
- virtual char \* [ButtonLabel](#) (int bt)  
*Return the label for the specified MFD button.*
- virtual int [ButtonMenu](#) (const MFDBUTTONMENU \*\*menu) const  
*Defines a list of short descriptions for the various MFD mode button/key functions.*
- virtual void [WriteStatus](#) (FILEHANDLE scn) const  
*Called when the MFD should write its status to a scenario file.*
- virtual void [ReadStatus](#) (FILEHANDLE scn)  
*Called when the MFD should read its status from a scenario file.*
- virtual void [StoreStatus](#) () const
- virtual void [RecallStatus](#) ()

### Protected Attributes

- DWORD **W**
- DWORD **H**  
*width and height of MFD display area (pixel)*
- DWORD **cw**
- DWORD **ch**  
*character width, height of standard MFD font 0 (pixel)*
- **VESSEL \* pV**  
*pointer to vessel interface*

### Friends

- class **MFD2**
- class **Instrument\_User**

### 16.36.2 Constructor & Destructor Documentation

#### 16.36.2.1 MFD::MFD (DWORD *w*, DWORD *h*, VESSEL \* *vessel*)

Constructor. Creates a new MFD.

##### Parameters:

- w** width of the MFD display (pixel)
- h** height of the MFD display (pixel)
- vessel** pointer to VESSEL interface associated with the MFD.

**Note:**

[MFD](#) is a pure virtual function, so it can't be instantiated directly. It is used as a base class for specialised [MFD](#) modes.

New [MFD](#) modes are registered by a call to [oapiRegisterMFDMode\(\)](#). Whenever the new mode is selected by the user, Orbiter sends a [OAPI\\_MSG\\_MFD\\_OPENED](#) signal to the message handler, to which the module should respond by creating the [MFD](#) mode and returning a pointer to it. Orbiter will automatically destroy the [MFD](#) mode when it is turned off.

### 16.36.3 Member Function Documentation

#### 16.36.3.1 [virtual void MFD::Update \(HDC hDC\)](#) [pure virtual]

Callback function: Orbiter calls this method when the [MFD](#) needs to update its display.

**Parameters:**

*hDC* Windows device context for drawing on the [MFD](#) display surface.

**Note:**

The frequency at which this function is called corresponds to the "MFD refresh rate" setting in Orbiter's parameter settings, unless a redraw is forced by [InvalidateDisplay\(\)](#).

**Deprecated**

This method is deprecated. MFD implementations should derive from [MFD2](#) and use the device-independent [MFD2::Update\(oapi::Sketchpad\\*\)](#) method instead.

Implemented in [MFD2](#).

#### 16.36.3.2 [void MFD::InvalidateDisplay \(\)](#)

Force a display update in the next frame.

Force a display update in the next frame. This function causes Orbiter to call the MFD's Update method in the next frame.

#### 16.36.3.3 [void MFD::InvalidateButtons \(\)](#)

Force the [MFD](#) buttons to be redrawn.

Force the [MFD](#) buttons to be redrawn. This is useful to alert Orbiter that the [MFD](#) mode has dynamically modified its button labels.

**Note:**

Orbiter will call the [MFD::ButtonLabel](#) method to retrieve the new button labels. Therefore this must have been updated to return the new labels before calling [InvalidateButtons\(\)](#).

If the [MFD](#) is part of a 2-D panel view or 3-D virtual cockpit view, Orbiter calls the [VES-SEL2::clbkMFDMode\(\)](#) method to allow the vessel to update its button labels. If the [MFD](#) is one of the two glass cockpit [MFD](#) displays, the buttons are updated internally.

If the [MFD](#) is displayed in an external window, Orbiter calls the [ExternMFD::clbkRefreshButtons\(\)](#) method to refresh the buttons.

**16.36.3.4 void MFD::Title (HDC *hDC*, const char \* *title*) const**

Displays a title string in the upper left corner of the [MFD](#) display.

**Parameters:**

*hDC* device context

*title* title string (null-terminated)

**Note:**

This method should be called from within [Update\(\)](#)

The title string can contain up to approx. 35 characters when displayed in the default Courier [MFD](#) font.

This method switches the text colour of the GDI context to white.

**Deprecated**

This method is deprecated. MFD implementations should derive from [MFD2](#) and use the device-independent [MFD2::Title](#) method instead.

**16.36.3.5 HPEN MFD::SelectDefaultPen (HDC *hDC*, DWORD *i*) const**

Selects a predefined pen into the device context.

**Parameters:**

*hDC* Windows device context

*i* pen index

**Returns:**

Handle of pen being replaced.

**Note:**

Currently supported are pen indices 0-5, where

0 = solid, HUD display colour

1 = solid, light green

2 = solid, medium green

3 = solid, medium yellow

4 = solid, dark yellow

5 = solid, medium grey

In principle, an [MFD](#) mode may create its own pen resources using the standard Windows CreatePen function, but using predefined pens is preferred to provide a consistent [MFD](#) look.

**Deprecated**

This method is deprecated. MFD implementations should derive from [MFD2](#) and use the device-independent [MFD2::GetDefaultPen](#) method instead.

**16.36.3.6 HFONT MFD::SelectDefaultFont (HDC *hDC*, DWORD *i*) const**

Selects a predefined [MFD](#) font into the device context.

**Parameters:**

*hDC* Windows device context

*i* font index

**Returns:**

Handle of font being replaced.

**Note:**

Currently supported are font indices 0-3, where

0 = standard MFD font (Courier, fixed pitch)

1 = small font (Arial, variable pitch)

2 = small font, rotated 90 degrees (Arial, variable pitch)

3 = medium font, (Arial, variable pitch)

In principle, an MFD mode may create its own fonts using the standard Windows CreateFont function, but using the predefined fonts is preferred to provide a consistent MFD look.

Default fonts are scaled automatically according to the MFD display size.

**Deprecated**

This method is deprecated. MFD implementations should derive from [MFD2](#) and use the device-independent [MFD2::GetDefaultFont](#) method instead.

**16.36.3.7 virtual bool MFD::ConsumeKeyBuffered (DWORD *key*) [inline, virtual]**

MFD keyboard handler for buffered keys.

**Parameters:**

*key* key code (*see OAPI\_KEY\_xxx constants in orbitersdk.h*)

**Returns:**

The function should return true if it recognises and processes the key, false otherwise.

**16.36.3.8 virtual bool MFD::ConsumeKeyImmediate (char \* *kstate*) [inline, virtual]**

MFD keyboard handler for immediate (unbuffered) keys.

**Parameters:**

*kstate* keyboard state.

**Returns:**

The function should return true only if it wants to inhibit Orbiter's default immediate key handler for this time step completely.

**Note:**

The state of single keys can be queried by the KEYDOWN macro.

The immediate key handler is useful where an action should take place while a key is pressed.

**16.36.3.9 virtual bool MFD::ConsumeButton (int *bt*, int *event*) [inline, virtual]**

MFD button handler.

MFD button handler. This function is called when the user performs a mouse click on a panel button associated with the MFD.

**Parameters:**

*bt* button identifier.

*event* mouse event (*see PANEL\_MOUSE\_xxx constants in orbitersdk.h*)

**Returns:**

The function should return true if it processes the button event, false otherwise.

**Note:**

This function is invoked as a response to a call to [oapiProcessMFDButton\(\)](#) in a vessel module.

Typically, [ConsumeButton\(\)](#) will call [ConsumeKeyBuffered\(\)](#) or [ConsumeKeyImmediate\(\)](#) to emulate a keyboard event.

**16.36.3.10 virtual char\* MFD::ButtonLabel (int *bt*) [inline, virtual]**

Return the label for the specified MFD button.

**Parameters:**

*bt* button identifier

**Returns:**

The function should return a 0-terminated character string of up to 3 characters, or NULL if the button is unlabelled.

**Bug**

This function should really return a const char\*

**16.36.3.11 virtual int MFD::ButtonMenu (const MFDBUTTONMENU \*\* *menu*) const [inline, virtual]**

Defines a list of short descriptions for the various MFD mode button/key functions.

**Parameters:**

*menu* on return this should point to an array of button menu items. (see notes)

**Returns:**

number of items in the list

**Note:**

The definition of the MFDBUTTONMENU struct is:

```
typedef struct {
    const char *line1, *line2;
```

```

    char selchar;
} MFDBUTTONMENU;
containing up to 2 lines of short description, and the keyboard key to trigger the function.
Each line should contain no more than 16 characters, to fit into the MFD display.
If the menu item only uses one line, then line2 should be set to NULL.
menu==0 is valid and indicates that the caller only requires the number of items, not the actual list.
A typical implementation would be

```

```

int MyMFD::ButtonMenu (const MFDBUTTONMENU **menu) const
{
    static const MFDBUTTONMENU mnu[2] = {
        {"Select target", 0, 'T'},
        {"Select orbit", "reference", 'R'}
    };
    if (menu) *menu = mnu;
    return 2;
}

```

#### **16.36.3.12 virtual void MFD::WriteStatus (FILEHANDLE *scn*) const [inline, virtual]**

Called when the **MFD** should write its status to a scenario file.

**Parameters:**

*scn* scenario file handle (write only)

**Note:**

Use the oapiWriteScenario\_xxx functions to write **MFD** status parameters to the scenario.  
The default behaviour is to do nothing. **MFD** modes which need to save status parameters should overwrite this function.

#### **16.36.3.13 virtual void MFD::ReadStatus (FILEHANDLE *scn*) [inline, virtual]**

Called when the **MFD** should read its status from a scenario file.

**Parameters:**

*scn* scenario file handle (read only)

**Note:**

Use a loop with [oapiReadScenario\\_nextline\(\)](#) to read **MFD** status parameters from the scenario.  
The default behaviour is to do nothing. **MFD** modes which need to read status parameters should overwrite this function.

#### **16.36.3.14 virtual void MFD::StoreStatus () const [inline, virtual]**

Called before destruction of the **MFD** mode, to allow the mode to save its status to static memory.

**Note:**

This function is called before an **MFD** mode is destroyed (either because the **MFD** switches to a different mode, or because the **MFD** itself is destroyed). It allows the **MFD** to back up its status parameters, so it can restore its last status when it is created next time.

Since the [MFD](#) mode instance is about to be destroyed, status parameters should be backed up either in static data members, or outside the class instance.

In principle this function could be implemented by opening a file and calling [WriteStatus\(\)](#) with the file handle. However for performance reasons file I/O should be avoided in this function.

The default behaviour is to do nothing. [MFD](#) modes which need to save status parameters should overwrite this function.

#### 16.36.3.15 virtual void MFD::RecallStatus () [inline, virtual]

Called after creation of the [MFD](#) mode, to allow the mode to restore its status from the last save.

**Note:**

This is the counterpart to the [StoreStatus\(\)](#) function. It should be implemented if and only if [StoreStatus\(\)](#) is implemented.

The documentation for this class was generated from the following file:

- Orbitersdk/include/[MFDAPIL.h](#)

## 16.37 MFD2 Class Reference

```
#include <MFDAPIL.h>
```

Inheritance diagram for MFD2:

Collaboration diagram for MFD2:

### 16.37.1 Detailed Description

Extended [MFD](#) class.

[MFD2](#) replaces GDI-specific functions with versions that use the generic Sketchpad class. [MFD](#) addons should derive from [MFD2](#) instead of [MFD](#), to ensure compatibility with non-GDI graphics clients.

## Public Member Functions

- **MFD2** (DWORD *w*, DWORD *h*, VESSEL \**vessel*)  
*Constructor: Creates a new MFD.*
- **~MFD2** ()  
*MFD destructor.*
- DWORD **GetWidth** () const  
*Returns the MFD display width.*
- DWORD **GetHeight** () const  
*Returns the MFD display height.*
- void **Update** (HDC *hDC*)  
*Dummy implementation of GDI-specific base class method.*
- virtual bool **Update** (oapi::Sketchpad \**skp*)  
*Callback function: Orbiter calls this method when the MFD needs to update its display.*
- void **Title** (oapi::Sketchpad \**skp*, const char \**title*) const  
*Displays a title string in the upper left corner of the MFD display.*
- oapi::Pen \* **GetDefaultPen** (DWORD *colidx*, DWORD *intens*=0, DWORD *style*=1) const  
*Returns a predefined MFD pen resource.*
- oapi::Font \* **GetDefaultFont** (DWORD *fontid*) const  
*Returns a predefined MFD font resource.*
- DWORD **GetDefaultColour** (DWORD *colidx*, DWORD *intens*=0) const  
*Returns the colour value for a specified colour index and intensity.*

### 16.37.2 Constructor & Destructor Documentation

#### 16.37.2.1 MFD2::MFD2 (DWORD *w*, DWORD *h*, VESSEL \**vessel*) [inline]

Constructor. Creates a new MFD.

##### Parameters:

- w* width of the MFD display (pixel)
- h* height of the MFD display (pixel)
- vessel* pointer to VESSEL interface associated with the MFD.

##### Note:

MFD is a pure virtual function, so it can't be instantiated directly. It is used as a base class for specialised MFD modes.

New MFD modes are registered by a call to `oapiRegisterMFDMode()`. Whenever the new mode is selected by the user, Orbiter sends a OAPI\_MSG\_MFD\_OPENED signal to the message handler, to which the module should respond by creating the MFD mode and returning a pointer to it. Orbiter will automatically destroy the MFD mode when it is turned off.

### 16.37.3 Member Function Documentation

#### 16.37.3.1 **DWORD MFD2::GetWidth () const [inline]**

Returns the [MFD](#) display width.

##### Returns:

[MFD](#) display width [pixel]

##### See also:

[GetHeight](#)

#### 16.37.3.2 **DWORD MFD2::GetHeight () const [inline]**

Returns the [MFD](#) display height.

##### Returns:

[MFD](#) display height [pixel]

##### See also:

[GetWidth](#)

#### 16.37.3.3 **void MFD2::Update (HDC *hDC*) [inline, virtual]**

Dummy implementation of GDI-specific base class method.

##### Note:

Derived classes should overload the [Update\(oapi::Sketchpad\\*\)](#) method instead.

Implements [MFD](#).

#### 16.37.3.4 **virtual bool MFD2::Update (oapi::Sketchpad \* *skp*) [virtual]**

Callback function: Orbiter calls this method when the [MFD](#) needs to update its display.

##### Parameters:

*skp* Drawing context for drawing on the [MFD](#) display surface.

##### Note:

The frequency at which this function is called corresponds to the "MFD refresh rate" setting in Orbiter's parameter settings, unless a redraw is forced by [InvalidateDisplay\(\)](#).

This function must be overwritten by derived classes.

**16.37.3.5 void MFD2::Title (oapi::Sketchpad \* *skp*, const char \* *title*) const**

Displays a title string in the upper left corner of the **MFD** display.

**Parameters:**

*skp* Drawing context

*title* title string (null-terminated)

**Note:**

This method should be called from within [Update\(\)](#)

The title string can contain up to approx. 35 characters when displayed in the default Courier **MFD** font.

This method switches the text colour of the drawing context to white.

**16.37.3.6 oapi::Pen\* MFD2::GetDefaultPen (DWORD *colidx*, DWORD *intens* = 0, DWORD *style* = 1) const**

Returns a predefined **MFD** pen resource.

**Parameters:**

*colidx* pen colour index (see notes)

*intens* pen brightness (0=bright, 1=dark)

*style* pen style (1=solid, 2=dashed)

**Returns:**

pen resource

**Note:**

Valid colour indices are 0 to 4, where

Index	Function	default colour
0	Main <b>MFD</b> colour	green
1	Auxiliary colour 1	yellow
2	Auxiliary colour 2	white
3	Auxiliary colour 3	red
4	Auxiliary colour 4	blue

To select the pen for drawing in the **MFD** display, call the **MFD** drawing context's [oapi::Sketchpad::SetPen](#) method.

The default colours can be overridden by editing Config/MFD/default.cfg.

In principle, an **MFD** mode may create its own pen resources using the [oapi::GraphicsClient::clbkCreatePen](#) function, but using predefined pens is preferred to provide a consistent **MFD** look, and to avoid excessive allocation of drawing resources.

**See also:**

[oapi::Sketchpad::SetPen](#)

**16.37.3.7 oapi::Font\* MFD2::GetDefaultFont (DWORD *fontidx*) const**

Returns a predefined [MFD](#) font resource.

**Parameters:**

*fontidx* font index

**Returns:**

font resource

**Note:**

Currently supported are font indices 0-2, where  
 0 = standard [MFD](#) font (*Courier*, fixed pitch)  
 1 = small font (*Arial*, variable pitch)  
 2 = small font, rotated 90 degrees (*Arial*, variable pitch)

To select the font for drawing in the [MFD](#) display, call the [MFD](#) drawing context's [oapi::Sketchpad::SetFont](#) method.

In principle, an [MFD](#) mode may create its own fonts using the standard Windows `CreateFont` function, but using the predefined fonts is preferred to provide a consistent [MFD](#) look.

Default fonts are scaled automatically according to the [MFD](#) display size.

**16.37.3.8 DWORD MFD2::GetDefaultColour (DWORD *colidx*, DWORD *intens* = 0) const**

Returns the colour value for a specified colour index and intensity.

**Parameters:**

*colidx* colour index (see notes)

*intens* colour brightness (0=bright, 1=dark)

**Returns:**

Colour value in 0xBBGGRR format.

**Note:**

Valid colour indices are 0 to 4, where

Index	Function	default colour
0	Main <a href="#">MFD</a> colour	green
1	Auxiliary colour 1	yellow
2	Auxiliary colour 2	white
3	Auxiliary colour 3	red
4	Auxiliary colour 4	blue

The returned colour values can be used to set standard text, pen or brush colours for particular display elements.

**See also:**

[oapi::Sketchpad::SetTextColor](#)

The documentation for this class was generated from the following file:

- Orbitersdk/include/[MFD API.h](#)

## 16.38 oapi::Module Class Reference

```
#include <ModuleAPI.h>
```

Inheritance diagram for oapi::Module:

Collaboration diagram for oapi::Module:

### 16.38.1 Detailed Description

Generic Orbiter plugin interface class.

Defines generic base class which can be used by plugins to provide a set of interface functions to the Orbiter core, and callback functions which can be overloaded by derived classes to react to specific types of events.

#### Public Types

- enum [RenderMode](#) { [RENDER\\_NONE](#), [RENDER\\_FULLSCREEN](#), [RENDER\\_WINDOW](#) }
- Simulation graphics support type.*

#### Public Member Functions

- [Module](#) (HINSTANCE hDLL)  
*Creates a new [Module](#) instance.*
- virtual void [clbkSimulationStart](#) ([RenderMode](#) mode)  
*Simulation start notification.*
- virtual void [clbkSimulationEnd](#) ()  
*Simulation end notification.*
- virtual void [clbkPreStep](#) (double simt, double simdt, double mjd)  
*Time step notification before state update.*

- virtual void [clbkPostStep](#) (double simt, double simdt, double mjd)  
*Time step notification after state update.*
- virtual void [clbkTimeJump](#) (double simt, double simdt, double mjd)  
*Discontinuous simulation time jump notification.*
- virtual void [clbkFocusChanged](#) ([OBJHANDLE](#) new\_focus, [OBJHANDLE](#) old\_focus)  
*Change of input focus notification.*
- virtual void [clbkTimeAccChanged](#) (double new\_warp, double old\_warp)  
*Change of time acceleration notification.*
- virtual void [clbkNewVessel](#) ([OBJHANDLE](#) hVessel)  
*Vessel creation notification.*
- virtual void [clbkDeleteVessel](#) ([OBJHANDLE](#) hVessel)  
*Vessel destruction notification.*
- virtual void [clbkVesselJump](#) ([OBJHANDLE](#) hVessel)  
*Discontinuous vessel repositioning notification.*
- virtual void [clbkPause](#) (bool pause)  
*Simulation pause/resume notification.*

## 16.38.2 Member Enumeration Documentation

### 16.38.2.1 enum oapi::Module::RenderMode

Simulation graphics support type.

See also:

[clbkSimulationStarted](#)

Enumerator:

***RENDER\_NONE*** no graphics support

***RENDER\_FULLSCREEN*** fullscreen mode

***RENDER\_WINDOW*** windowed mode

## 16.38.3 Constructor & Destructor Documentation

### 16.38.3.1 oapi::Module::Module (HINSTANCE *hDLL*)

Creates a new [Module](#) instance.

Parameters:

***hDLL*** DLL library instance handle (see [InitModule](#))

#### 16.38.4 Member Function Documentation

##### 16.38.4.1 virtual void oapi::Module::clbkSimulationStart (RenderMode mode) [virtual]

Simulation start notification.

This method is called immediately after a simulation session has been set up (i.e. all objects created and their states set according to the scenario data) and the render window has been opened (if applicable).

**Parameters:**

*mode* defines the graphics support (none, fullscreen or windowed)

**Default action:**

Calls [opcOpenRenderViewport](#), if defined in the module.

##### 16.38.4.2 virtual void oapi::Module::clbkSimulationEnd () [virtual]

Simulation end notification.

This method is called immediately before a simulation session is terminated, and before the render window is closed.

**Default action:**

Calls [opcCloseRenderViewport](#), if defined in the module.

##### 16.38.4.3 virtual void oapi::Module::clbkPreStep (double simt, double simdt, double mjd) [virtual]

Time step notification before state update.

Called at each time step of the simulation, before the state is updated to the current simulation time. This function is only called when the "physical" state of the simulation is propagated in time. clbkPreStep is not called while the simulation is paused, even if the user moves the camera.

**Parameters:**

*simt* simulation time after the currently processed step [s]

*simdt* length of the currently processed step [s]

*mjd* simulation time after the currently processed step in Modified Julian Date format [days]

**Default action:**

Calls [opcPreStep](#), if defined in the module.

**Note:**

This function is called by Orbiter after the new time step length (simdt) and simulation time (simt) have been calculated, but before the simulation state is integrated to simt. The parameters passed to clbkPreStep therefore are the values that will be applied in the current simulation step.

**See also:**

[clbkPostStep](#)

**16.38.4.4 virtual void oapi::Module::clbkPostStep (double *simt*, double *simdt*, double *mjd*) [virtual]**

Time step notification after state update.

Called at each time step of the simulation, after the state has been updated to the current simulation time.

**Parameters:**

*simt* current simulation time [s]

*simdt* length of the last time step [s]

*mjd* simulation time in Modified Julian Date format [days]

**Default action:**

Calls [opcPostStep](#), if defined in the module.

**See also:**

[clbkPreStep](#)

**16.38.4.5 virtual void oapi::Module::clbkTimeJump (double *simt*, double *simdt*, double *mjd*) [inline, virtual]**

Discontinuous simulation time jump notification.

Called after a discontinuous explicit reset of the simulation time (e.g. using the scenario editor).

**Parameters:**

*simt* new simulation time relative to session start [s]

*simdt* jump interval [s]

*mjd* new absolute simulation time in MJD format [days]

**Default action:**

None.

**Note:**

*simdt* can be negative if a jump to an earlier time was performed.

*simt* can become negative if a jump prior to the session start time was performed.

**16.38.4.6 virtual void oapi::Module::clbkFocusChanged (OBJHANDLE *new\_focus*, OBJHANDLE *old\_focus*) [virtual]**

Change of input focus notification.

Called when input focus (keyboard and joystick control) is switched to a new vessel (for example as a result of a call to [oapiSetFocus](#)).

**Parameters:**

*new\_focus* handle of vessel receiving the input focus

*old\_focus* handle of vessel losing focus

**Default action:**

Calls [opcFocusChanged](#), if defined in the module.

**Note:**

Currently only objects of type "vessel" can receive the input focus. This may change in future versions. This callback function is also called at the beginning of the simulation, where new\_focus is the vessel receiving the initial focus, and old\_focus is NULL.

clbkFocusChanged is sent to non-vessel modules after the vessels receiving and losing focus have been notified via [VESSEL2::clbkFocusChanged](#).

**16.38.4.7 virtual void oapi::Module::clbkTimeAccChanged (double *new\_warp*, double *old\_warp*) [virtual]**

Change of time acceleration notification.

Called when the simulation time acceleration factor changes.

**Parameters:**

*new\_warp* new time acceleration factor

*old\_warp* old time acceleration factor

**Default action:**

Calls [opcTimeAccChanged](#), if defined in the module.

**16.38.4.8 virtual void oapi::Module::clbkNewVessel (OBJHANDLE *hVessel*) [inline, virtual]**

Vessel creation notification.

Sent to modules after a new vessel has been created during the simulation run. Not sent for vessels created from the scenario script at the start of a session.

**Parameters:**

*hVessel* object handle for the new vessel

**Default action:**

None.

**16.38.4.9 virtual void oapi::Module::clbkDeleteVessel (OBJHANDLE *hVessel*) [virtual]**

Vessel destruction notification.

Sent to modules immediately before a vessel is destroyed. After this callback method returns, the object handle (*hVessel*) and will no longer be valid. Modules should make sure that they don't access the vessel in any form after this point.

**Parameters:**

*hVessel* object handle for the vessel being destroyed.

**Default action:**

Calls [opcDeleteVessel](#), if defined in the module.

**16.38.4.10 virtual void oapi::Module::clbkVesselJump (OBJHANDLE *hVessel*) [inline, virtual]**

Discontinuous vessel repositioning notification.

Sent to modules after a vessel position has been set explicitly (rather than via continuous state propagation). This callback can be used to force a refresh of parameters that depend on vessel position.

**Parameters:**

*hVessel* vessel object handle

**Default action:**

None.

**Note:**

This method is called after a [VESSEL::ShiftCentreOfMass\(\)](#)

**16.38.4.11 virtual void oapi::Module::clbkPause (bool *pause*) [virtual]**

Simulation pause/resume notification.

Called when the pause/resume state of the simulation has changed.

**Parameters:**

*pause* pause/resume state: true if simulation has been paused, false if simulation has been resumed.

**Default action:**

Calls [opcPause](#), if defined in the module.

The documentation for this class was generated from the following file:

- Orbitersdk/include/ModuleAPI.h

## 16.39 oapi::ModuleNV Class Reference

```
#include <ModuleAPI.h>
```

Inheritance diagram for oapi::ModuleNV:

### 16.39.1 Detailed Description

Generic Orbiter plugin interface class.

Defines generic base class which can be used by plugins to provide a set of interface functions to the Orbiter core. This class contains only the non-virtual set of methods (excluding callback functions). Plugin implementations should normally not derive their interface classes from [oapi::ModuleNV](#), but instead from class [oapi::Module](#) which includes the virtual callback methods.

#### Public Member Functions

- [ModuleNV \(HINSTANCE hDLL\)](#)  
*Creates a new [ModuleNV](#) instance.*
- int [Version \(\) const](#)  
*[Module](#) interface version.*
- HINSTANCE [GetModule \(\) const](#)  
*Returns the module instance handle.*
- double [GetSimTime \(\) const](#)  
*Returns simulation time since session start.*
- double [GetSimStep \(\) const](#)  
*Returns the length of the last time step.*
- double [GetSimMJD \(\) const](#)  
*Returns the absolute simulation time in Modified Julian Date format.*

#### Protected Attributes

- int **version**
- HINSTANCE **hModule**

### 16.39.2 Constructor & Destructor Documentation

#### 16.39.2.1 oapi::ModuleNV::ModuleNV (HINSTANCE *hDLL*)

Creates a new [ModuleNV](#) instance.

##### Parameters:

*hDLL* DLL library instance handle (see [InitModule](#))

### 16.39.3 Member Function Documentation

#### 16.39.3.1 int oapi::ModuleNV::Version () const [inline]

[Module](#) interface version.

##### Returns:

version number

**16.39.3.2 HINSTANCE oapi::ModuleNV::GetModule () const [inline]**

Returns the module instance handle.

**Returns:**

[Module](#) instance handle.

**16.39.3.3 double oapi::ModuleNV::GetSimTime () const**

Returns simulation time since session start.

**Returns:**

Simulation session time [s]

**Note:**

The simulation session time is useful mainly for time differences. To get an absolute time parameter, use [GetSimMJD](#).

**See also:**

[GetSimMJD](#), [GetSimStep](#)

**16.39.3.4 double oapi::ModuleNV::GetSimStep () const**

Returns the length of the last time step.

**Returns:**

Step length [s]

**Note:**

This method returns the time difference between the current and previous time frame.  
This parameter is useful for numerical (finite difference) calculation of time derivatives.

**See also:**

[GetSimTime](#)

**16.39.3.5 double oapi::ModuleNV::GetSimMJD () const**

Returns the absolute simulation time in Modified Julian Date format.

**Returns:**

Current Modified Julian Date [days]

**Note:**

Orbiter defines the Modified Julian Date (MJD) as JD - 2 400 000.5, where JD is the Julian Date. JD is the interval of time in mean solar days elapsed since 4713 BC January 1 at Greenwich mean noon.

**See also:**[GetSimTime](#)

The documentation for this class was generated from the following file:

- Orbitersdk/include/ModuleAPI.h

## 16.40 NAVDATA Struct Reference

```
#include <OrbiterAPI.h>
```

### 16.40.1 Detailed Description

Navigation transmitter data.

This structure contains both general data (transmitter type, channel, output power and description string) and type-specific data. To query type-specific data, first check the transmitter type, for example

```
NAVDATA ndata;
oapiGetNavData (hNav, &ndata);
if (ndata.type == TRANSMITTER_ILS)
    approach_dir = ndata.ils.appdir;
```

**Note:**

The power  $S_0$  of a transmitter is defined in arbitrary units such that the signal  $S(r) = S_0 / r^2$  drops to 1 at the maximum range  $r_{max}$ , given a default receiver, i.e.  $S_0 = r_{max}^2$ .

**See also:**[oapiGetNavData](#)

### Public Attributes

- DWORD **type**  
*transmitter type id*
- DWORD **ch**  
*transmitter channel (0..639)*
- double **power**  
*transmitter power [arbitrary units]*
- const char \* **descr**  
*pointer to transmitter description string*
- union {
 struct {
 OBJHANDLE hPlanet
 *associated planet*
 double lng
 double lat
 }
 }

```

transmitter location [rad]
} vor
struct {
    OBJHANDLE hBase
    associated base
    int npad
    pad number (>= 0)
} vtol
struct {
    OBJHANDLE hBase
    associated base
    double appdir
    ILS approach direction [rad].
} ils
struct {
    OBJHANDLE hVessel
    associated vessel
    DOCKHANDLE hDock
    associated docking port
} ids
struct {
    OBJHANDLE hVessel
    associated vessel
} xpdr
};
```

The documentation for this struct was generated from the following file:

- OrbiterSDK/include/OrbiterAPI.h

## 16.41 NTVERTEX Struct Reference

```
#include <OrbiterAPI.h>
```

### 16.41.1 Detailed Description

vertex definition including normals and texture coordinates

#### Public Attributes

- float **x**  
*vertex x position*
- float **y**  
*vertex y position*
- float **z**  
*vertex z position*

- float **nx**  
*vertex x normal*
- float **ny**  
*vertex y normal*
- float **nz**  
*vertex z normal*
- float **tu**  
*vertex u texture coordinate*
- float **tv**  
*vertex v texture coordinate*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 16.42 ORBITPARAM Struct Reference

```
#include <OrbiterAPI.h>
```

### 16.42.1 Detailed Description

Secondary orbital parameters derived from the primary [ELEMENTS](#).

This members of this structure provide additional parameters to the primary elements of contained in the [ELEMENTS](#) structure.

**Note:**

SMi: for open orbits, this represents the imaginary semi-axis  
PeD: distance to lowest point of the orbit from focal point  
ApD: distance of highest point of the orbit from focal point. Only defined for closed orbits.  
T: orbit period only defined for closed orbits.  
PeT: For open orbits, this is negative after periapsis passage  
ApT: Only defined for closed orbits.

**See also:**

[ELEMENTS](#), Basics of orbital mechanics

### Public Attributes

- double **SMi**  
*semi-minor axis [m]*
- double **PeD**  
*periapsis distance [m]*

- double [ApD](#)  
*apoapsis distance [m]*
- double [MnA](#)  
*mean anomaly [rad]*
- double [TrA](#)  
*true anomaly [rad]*
- double [MnL](#)  
*mean longitude [rad]*
- double [TrL](#)  
*true longitude [rad]*
- double [EcA](#)  
*eccentric anomaly [rad]*
- double [Lec](#)  
*linear eccentricity [m]*
- double [T](#)  
*orbit period [s]*
- double [PeT](#)  
*time to next periapsis passage [s]*
- double [ApT](#)  
*time to next apoapsis passage [s]*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 16.43 oapi::ParticleStream Class Reference

```
#include <GraphicsAPI.h>
```

Collaboration diagram for oapi::ParticleStream:

### 16.43.1 Detailed Description

Defines an array of "particles" (e.g. for exhaust and reentry effects, gas venting, condensation, etc.).

Each particle is represented by a "billboard" object facing the camera and rendered with a semi-transparent texture.

Particle streams experience drag in atmosphere. They also can cast shadows on the ground.

#### Public Member Functions

- **ParticleStream** (`GraphicsClient *_gc`, `PARTICLESTREAMSPEC *pss`)  
*Constructs a new particle stream.*
- **~ParticleStream** ()  
*Destructor.*
- **void Attach** (`OBJHANDLE hObj`, const `VECTOR3 *ppos`, const `VECTOR3 *pdir`, const double `*srclvl`)  
*Attach the stream to an object.*
- **void Attach** (`OBJHANDLE hObj`, const `VECTOR3 &_pos`, const `VECTOR3 &_dir`, const double `*srclvl`)  
*Attach the stream to an object.*
- **void Detach** ()  
*Detach the stream from its object.*
- **void SetFixedPos** (const `VECTOR3 &_pos`)  
*Reset the particle source point to a constant value.*
- **void SetFixedDir** (const `VECTOR3 &_dir`)  
*Reset the particle source direction to a constant value.*
- **void SetVariablePos** (const `VECTOR3 *ppos`)

*Reset the particle source point reference.*

- void **SetVariableDir** (const **VECTOR3** \**pdir*)  
*Reset the particle source direction reference.*
- void **SetLevelPtr** (const double \**srclvl*)  
*Reset the particle generator strength reference.*
- const double \* **Level** () const  
*Returns the particle generator level.*

### Protected Attributes

- **GraphicsClient** \* **gc**
- const double \* **level**
- **OBJHANDLE** **hRef**
- const **VECTOR3** \* **pos**
- const **VECTOR3** \* **dir**
- **VECTOR3** **lpos**
- **VECTOR3** **ldir**

### Friends

- class **GraphicsClient**

#### 16.43.2 Constructor & Destructor Documentation

##### 16.43.2.1 oapi::ParticleStream::ParticleStream (**GraphicsClient** \* *\_gc*, **PARTICLESTREAM-SPEC** \**pss*)

Constructs a new particle stream.

###### Parameters:

*\_gc* pointer to graphics client  
*pss* particle parameter set

###### Note:

The particle stream will only start to generate particles once it has been attached to an object with [Attach\(\)](#).

#### 16.43.3 Member Function Documentation

##### 16.43.3.1 void oapi::ParticleStream::Attach (**OBJHANDLE** *hObj*, const **VECTOR3** \**ppos*, const **VECTOR3** \**pdir*, const double \**srclvl*)

Attach the stream to an object.

###### Parameters:

*hObj* object handle

*ppos* pointer to particle source point (object frame)

*pdir* pointer to particle direction (object frame)

*srclvl* pointer to particle generator level

**Note:**

This method uses pointers to externally defined position and direction variables which may be modified by orbiter during the lifetime of the particle stream.

**16.43.3.2 void oapi::ParticleStream::Attach (OBJHANDLE *hObj*, const VECTOR3 & *\_pos*, const VECTOR3 & *\_dir*, const double \* *srclvl*)**

Attach the stream to an object.

**Parameters:**

*hObj* object handle

*\_pos* particle source point (object frame)

*\_dir* particle direction (object frame)

*srclvl* pointer to particle generator level

**Note:**

This method uses fixed position and direction variables.

**16.43.3.3 void oapi::ParticleStream::Detach ()**

Detach the stream from its object.

**Note:**

After detaching the stream, no new particles will be generated, but the existing particle will persist to the end of their lifetime.

**16.43.3.4 void oapi::ParticleStream::SetFixedPos (const VECTOR3 & *\_pos*)**

Reset the particle source point to a constant value.

**Parameters:**

*\_pos* particle source point (reference object frame)

**Note:**

This method overrides any previous fixed or variable source position.

**16.43.3.5 void oapi::ParticleStream::SetFixedDir (const VECTOR3 & *\_dir*)**

Reset the particle source direction to a constant value.

**Parameters:**

*\_dir* particle direction (reference object frame)

**Note:**

This method overrides any previous fixed or variable source direction.

**16.43.3.6 void oapi::ParticleStream::SetVariablePos (const VECTOR3 \* *ppos*)**

Reset the particle source point reference.

**Parameters:**

*ppos* pointer to particle source point

**Note:**

This method overrides the previous position reference and any constant position value.

**16.43.3.7 void oapi::ParticleStream::SetVariableDir (const VECTOR3 \* *pdir*)**

Reset the particle source direction reference.

**Parameters:**

*pdir* pointer to particle source direction

**Note:**

This method overrides the previous direction reference and any constant direction value.

**16.43.3.8 void oapi::ParticleStream::SetLevelPtr (const double \* *srclvl*)**

Reset the particle generator strength reference.

**Parameters:**

*srclvl* pointer to particle generator strength (0...1)

**Note:**

The generator strength affects the initial opacity of generated particles.

This method overrides the previous strength reference.

**16.43.3.9 const double\* oapi::ParticleStream::Level () const [inline]**

Returns the particle generator level.

**Returns:**

pointer to particle generator level (0...1)

The documentation for this class was generated from the following file:

- Orbitersdk/include/GraphicsAPI.h

**16.44 PARTICLESTREAMSPEC Struct Reference**

```
#include <OrbiterAPI.h>
```

**16.44.1 Detailed Description**

Particle stream parameters.

**Note:**

The following mapping methods (LEVELMAP) between stream level L and opacity  $\alpha$  are supported:

- LVL\_FLAT:  $\alpha = \text{const}$
- LVL\_LIN:  $\alpha = L$
- LVL\_SQRT:  $\alpha = \sqrt{L}$
- LVL\_PLIN:  $\alpha = \begin{cases} 0 & \text{if } L < L_{\min} \\ \frac{L - L_{\min}}{L_{\max} - L_{\min}} & \text{if } L_{\min} \leq L \leq L_{\max} \\ 1 & \text{if } L > L_{\max} \end{cases}$
- LVL\_PSQRT:  $\alpha = \begin{cases} 0 & \text{if } L < L_{\min} \\ \sqrt{\frac{L - L_{\min}}{L_{\max} - L_{\min}}} & \text{if } L_{\min} \leq L \leq L_{\max} \\ 1 & \text{if } L > L_{\max} \end{cases}$

**Public Types**

- enum **LTYPE** { **EMISSIVE**, **DIFFUSE** }
- Particle lighting method.*
- enum **LEVELMAP** {
 **LVL\_FLAT**, **LVL\_LIN**, **LVL\_SQRT**, **LVL\_PLIN**,  
**LVL\_PSQRT** }
- Mapping from level to alpha value (particle opacity).*
- enum **ATMSMAP** { **ATM\_FLAT**, **ATM\_PLIN**, **ATM\_PLOG** }

### Public Attributes

- **DWORD flags**  
*streamspec bitflags*
- **double srcsize**  
*particle size at creation [m]*
- **double srcrate**  
*average particle creation rate [Hz]*
- **double v0**  
*emission velocity [m/s]*
- **double srcspread**  
*velocity spread during creation*
- **double lifetime**  
*average particle lifetime [s]*
- **double growthrate**  
*particle growth rate [m/s]*
- **double atmslowdown**  
*slowdown rate in atmosphere*
- **enum PARTICLESTREAMSPEC::LTYPE ltype**  
*Particle lighting method.*
- **enum PARTICLESTREAMSPEC::LEVELMAP levelmap**  
*Mapping from level to alpha value (particle opacity).*
- **double lmin**
- **double lmax**  
*min and max levels for level PLIN and PSQRT mapping types*
- **enum PARTICLESTREAMSPEC::ATMSMAP atmsmap**  
*mapping from atmospheric params to alpha*
- **double amin**
- **double amax**  
*min and max densities for atms PLIN mapping*
- **SURFHANDLE tex**  
*particle texture handle (NULL for default)*

### 16.44.2 Member Enumeration Documentation

#### 16.44.2.1 enum PARTICLESTREAMSPEC::LTYPE

Particle lighting method.

##### Enumerator:

*EMISSIVE* emissive lighting (example: plasma stream)

*DIFFUSE* diffuse lighting (example: vapour stream)

#### 16.44.2.2 enum PARTICLESTREAMSPEC::LEVELMAP

Mapping from level to alpha value (particle opacity).

##### Enumerator:

*LVL\_FLAT* constant (alpha independent of level)

*LVL\_LIN* linear mapping (alpha = level)

*LVL\_SQRT* square root mapping (alpha = sqrt(level))

*LVL\_PLIN* linear mapping in sub-range

*LVL\_PSQRT* square-root mapping in sub-range

### 16.44.3 Member Data Documentation

#### 16.44.3.1 enum PARTICLESTREAMSPEC::LTYPE PARTICLESTREAMSPEC::ltype

Particle lighting method.

render lighting method

#### 16.44.3.2 enum PARTICLESTREAMSPEC::LEVELMAP PARTICLESTREAMSPEC::levelmap

Mapping from level to alpha value (particle opacity).

mapping from level to alpha

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 16.45 oapi::Pen Class Reference

```
#include <DrawAPI.h>
```

Inheritance diagram for oapi::Pen:

Collaboration diagram for oapi::Pen:

### 16.45.1 Detailed Description

A pen is a resource used for drawing lines and the outlines of closed figures such as rectangles, ellipses and polygons.

#### Public Member Functions

- virtual `~Pen ()`  
*Pen destructor.*

#### Protected Member Functions

- `Pen (int style, int width, DWORD col)`  
*Pen constructor.*

### 16.45.2 Constructor & Destructor Documentation

#### 16.45.2.1 oapi::Pen::Pen (int *style*, int *width*, DWORD *col*) [inline, protected]

*Pen* constructor.

##### Parameters:

- style* line style (0=invisible, 1=solid, 2=dashed)
- width* line width [pixel]
- col* line colour (format: 0xBBGGRR)

The documentation for this class was generated from the following file:

- Orbitersdk/include/DrawAPI.h

## 16.46 PointLight Class Reference

```
#include <OrbiterAPI.h>
```

Inheritance diagram for PointLight:

Collaboration diagram for PointLight:

### 16.46.1 Detailed Description

Class for isotropic point light source.

#### Public Member Functions

- `PointLight (OBJHANDLE hObj, const VECTOR3 &_pos, double _range, double att0, double att1, double att2)`  
*Creates a white isotropic point light.*
- `PointLight (OBJHANDLE hObj, const VECTOR3 &_pos, double _range, double att0, double att1, double att2, COLOUR4 diffuse, COLOUR4 specular, COLOUR4 ambient)`  
*Creates a coloured isotropic point light.*
- `double GetRange () const`  
*Returns the light source range.*
- `void SetRange (double _range)`  
*Set the light source range.*
- `const double * GetAttenuation () const`  
*Returns a pointer to attenuation coefficients.*
- `void SetAttenuation (double att0, double att1, double att2)`

*Set the attenuation coefficients.*

### Protected Attributes

- double **range**
- double **att** [3]

### 16.46.2 Constructor & Destructor Documentation

#### 16.46.2.1 PointLight::PointLight (OBJHANDLE *hObj*, const VECTOR3 & *\_pos*, double *\_range*, double *att0*, double *att1*, double *att2*)

Creates a white isotropic point light.

##### Parameters:

- hObj* handle of object the point light is attached to  
*\_pos* light position in local object coordinates [m]  
*\_range* point light range [m]  
*att0* light attenuation parameters  
*att1* light attenuation parameters  
*att2* light attenuation parameters

#### 16.46.2.2 PointLight::PointLight (OBJHANDLE *hObj*, const VECTOR3 & *\_pos*, double *\_range*, double *att0*, double *att1*, double *att2*, COLOUR4 *diffuse*, COLOUR4 *specular*, COLOUR4 *ambient*)

Creates a coloured isotropic point light.

##### Parameters:

- hObj* handle of object the point light is attached to  
*\_pos* point light position in local object coordinates [m]  
*\_range* spotlight range [m]  
*att0* light attenuation parameters  
*att1* light attenuation parameters  
*att2* light attenuation parameters  
*diffuse* light source's contribution to lit objects' diffuse colour component  
*specular* light source's contribution to lit objects' specular colour component  
*ambient* light source's contribution to lit objects' ambient colour component

### 16.46.3 Member Function Documentation

#### 16.46.3.1 double PointLight::GetRange () const [inline]

Returns the light source range.

##### Returns:

Light source range [m]

**16.46.3.2 void PointLight::SetRange (double *\_range*)**

Set the light source range.

**Parameters:**

*\_range* new light source range [m]

**Note:**

When changing the range, the attenuation factors usually should be adjusted accordingly, to avoid sharp cutoff edges or large areas of negligible intensity.

**16.46.3.3 const double\* PointLight::GetAttenuation () const [inline]**

Returns a pointer to attenuation coefficients.

**Returns:**

Pointer to array of 3 attenuation coefficients.

**Note:**

The attenuation coefficients define the fractional light intensity  $I/I_0$  as a function of distance  $d$ :

$$\frac{I}{I_0} = \frac{1}{att_0 + datt_1 + d^2att_2}$$

**16.46.3.4 void PointLight::SetAttenuation (double *att0*, double *att1*, double *att2*)**

Set the attenuation coefficients.

**Parameters:**

*att0* attenuation coefficient  
*att1* attenuation coefficient  
*att2* attenuation coefficient

**Note:**

The attenuation coefficients define the fractional light intensity  $I/I_0$  as a function of distance  $d$ :

$$\frac{I}{I_0} = \frac{1}{att_0 + datt_1 + d^2att_2}$$

The documentation for this class was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

**16.47 oapi::ScreenAnnotation Class Reference**

```
#include <GraphicsAPI.h>
```

Collaboration diagram for oapi::ScreenAnnotation:

### 16.47.1 Detailed Description

Defines a block of text displayed on top of the simulation render window.

#### Public Member Functions

- **ScreenAnnotation (GraphicsClient \*\_gc)**  
*Constructs a new annotation object.*
- **virtual ~ScreenAnnotation ()**  
*Destroys the annotation object.*
- **virtual void Reset ()**  
*Resets annotation parameters to their default values.*
- **virtual void SetText (char \*str)**  
*Set the text to be displayed by the annotation object.*
- **virtual void ClearText ()**  
*Clear the text display.*
- **virtual void SetPosition (double x1, double y1, double x2, double y2)**  
*Set the bounding box of the annotation block.*
- **virtual void SetSize (double scale)**  
*Set the font size.*
- **virtual void SetColour (const VECTOR3 &col)**

*Set the font colour.*

- virtual void **Render** ()

*Render the annotation text into the simulation window.*

## 16.47.2 Constructor & Destructor Documentation

### 16.47.2.1 oapi::ScreenAnnotation::ScreenAnnotation (GraphicsClient \* *\_gc*)

Constructs a new annotation object.

#### Parameters:

*\_gc* pointer to graphics client

## 16.47.3 Member Function Documentation

### 16.47.3.1 virtual void oapi::ScreenAnnotation::SetText (char \* *str*) [virtual]

Set the text to be displayed by the annotation object.

#### Parameters:

*str* character string

### 16.47.3.2 virtual void oapi::ScreenAnnotation::SetPosition (double *x1*, double *y1*, double *x2*, double *y2*) [virtual]

Set the bounding box of the annotation block.

#### Parameters:

*x1* left edge

*y1* top edge

*x2* right edge

*y2* bottom edge

#### Note:

The positions are relative to the boundaries of the render window, in the range 0 to 1, where (0,0) is the top left edge, and (1,1) is the bottom right edge of the render window.

### 16.47.3.3 virtual void oapi::ScreenAnnotation::SetSize (double *scale*) [virtual]

Set the font size.

#### Parameters:

*scale* font size parameter (>0)

#### Note:

*scale*=1 defines the default font size, *scale*<1 is a smaller font, and *scale*>1 a larger font.  
The default font size is automatically scaled with the size of the render window.

### 16.47.3.4 virtual void oapi::ScreenAnnotation::SetColour (const VECTOR3 & col) [virtual]

Set the font colour.

**Parameters:**

*col* RGB values of the font colour (0..1 in each component)

The documentation for this class was generated from the following file:

- Orbitersdk/include/GraphicsAPI.h

## 16.48 oapi::Sketchpad Class Reference

```
#include <DrawAPI.h>
```

### 16.48.1 Detailed Description

A Sketchpad object defines an environment for drawing onto 2-D surfaces.

It defines drawing primitives (lines, text, etc.) that can be used for preparing MFD surfaces, panel elements or vessel markings.

The drawing object is an abstract class which must be implemented by derived graphics clients. An example for a DrawingObject implementation is via the Windows GDI (graphics device interface).

### Public Types

- enum **TAlign\_horizontal** { **LEFT**, **CENTER**, **RIGHT** }  
*Horizontal text alignment modes.*
- enum **TAlign\_vertical** { **TOP**, **BASELINE**, **BOTTOM** }  
*Vertical text alignment modes.*
- enum **BkgMode** { **BK\_TRANSPARENT**, **BK\_OPAQUE** }  
*Background modes for text output.*

### Public Member Functions

- **Sketchpad (SURFHANDLE s)**  
*Constructs a drawing object for a given surface.*
- virtual **~Sketchpad ()**  
*Destructor. Destroys a drawing object.*
- virtual **Font \* SetFont (Font \*font) const**  
*Selects a new font to use.*
- virtual **Pen \* SetPen (Pen \*pen) const**

*Selects a new pen to use.*

- virtual `Brush * SetBrush (Brush *brush) const`

*Selects a new brush to use.*

- virtual void `SetTextAlign (TAlign_horizontal tah=LEFT, TAlign_vertical tav=TOP)`

*Set horizontal and vertical text alignment.*

- virtual DWORD `SetTextColor (DWORD col)`

*Set the foreground colour for text output.*

- virtual DWORD `SetBackgroundColor (DWORD col)`

*Set the background colour for text output.*

- virtual void `SetBackgroundMode (BkgMode mode)`

*Set the background mode for text output.*

- virtual DWORD `GetCharSize ()`

*Return height and (average) width of a character in the currently selected font.*

- virtual DWORD `GetTextWidth (const char *str, int len=0)`

*Return the width of a text string in the currently selected font.*

- virtual void `SetOrigin (int x, int y)`

*Set the position in the surface bitmap which is mapped to the origin of the coordinate system for all drawing functions.*

- virtual bool `Text (int x, int y, const char *str, int len)`

*Draw a text string.*

- virtual bool `TextBox (int x1, int y1, int x2, int y2, const char *str, int len)`

*Draw a text string into a rectangle.*

- virtual void `Pixel (int x, int y, DWORD col)`

*Draw a single pixel in a specified colour.*

- virtual void `MoveTo (int x, int y)`

*Move the drawing reference to a new point.*

- virtual void `LineTo (int x, int y)`

*Draw a line to a specified point.*

- virtual void `Line (int x0, int y0, int x1, int y1)`

*Draw a line between two points.*

- virtual void `Rectangle (int x0, int y0, int x1, int y1)`

*Draw a rectangle (filled or outline).*

- virtual void `Ellipse (int x0, int y0, int x1, int y1)`

*Draw an ellipse from its bounding box.*

- virtual void [Polygon](#) (const IVECTOR2 \*pt, int npt)  
*Draw a closed polygon given by vertex points.*
- virtual void [Polyline](#) (const IVECTOR2 \*pt, int npt)  
*Draw a line of piecewise straight segments.*
- virtual void [PolyPolygon](#) (const IVECTOR2 \*pt, const int \*npt, const int nline)  
*Draw a set of polygons.*
- virtual void [PolyPolyline](#) (const IVECTOR2 \*pt, const int \*npt, const int nline)  
*Draw a set of polylines.*
- [SURFHANDLE GetSurface](#) () const  
*Returns the surface associated with the drawing object.*
- virtual HDC [GetDC](#) ()  
*Return the Windows device context handle, if applicable.*

## 16.48.2 Member Enumeration Documentation

### 16.48.2.1 enum oapi::Sketchpad::TAlign\_horizontal

Horizontal text alignment modes.

**See also:**

[SetTextAlign](#)

**Enumerator:**

- LEFT** align left  
**CENTER** align center  
**RIGHT** align right

### 16.48.2.2 enum oapi::Sketchpad::TAlign\_vertical

Vertical text alignment modes.

**See also:**

[SetTextAlign](#)

**Enumerator:**

- TOP** align top of text line  
**BASELINE** align base line of text line  
**BOTTOM** align bottom of text line

### 16.48.2.3 enum oapi::Sketchpad::BkgMode

Background modes for text output.

**See also:**

[SetBackgroundMode](#)

**Enumerator:**

*BK\_TRANSPARENT* transparent background

*BK\_OPAQUE* opaque background

## 16.48.3 Constructor & Destructor Documentation

### 16.48.3.1 oapi::Sketchpad::Sketchpad (SURFHANDLE *s*)

Constructs a drawing object for a given surface.

**Parameters:**

*s* surface handle

## 16.48.4 Member Function Documentation

### 16.48.4.1 virtual Font\* oapi::Sketchpad::SetFont (Font \**font*) const [inline, virtual]

Selects a new font to use.

**Parameters:**

*font* pointer to font resource

**Returns:**

Previously selected font.

**Default action:**

None, returns NULL.

**See also:**

[oapi::Font](#), [oapi::GraphicsClient::clbkCreateFont](#)

### 16.48.4.2 virtual Pen\* oapi::Sketchpad::SetPen (Pen \**pen*) const [inline, virtual]

Selects a new pen to use.

**Parameters:**

*pen* pointer to pen resource, or NULL to disable outlines

**Returns:**

Previously selected pen.

**Default action:**

None, returns NULL.

**See also:**

[oapi::Pen](#), [oapi::GraphicsClient::clbkCreatePen](#)

**16.48.4.3 virtual Brush\* oapi::Sketchpad::SetBrush (Brush \* brush) const [inline, virtual]**

Selects a new brush to use.

**Parameters:**

*brush* pointer to brush resource, or NULL to disable fill mode

**Returns:**

Previously selected brush.

**Default action:**

None, returns NULL.

**See also:**

[oapi::Brush](#), [oapi::GraphicsClient::clbkCreateBrush](#)

**16.48.4.4 virtual void oapi::Sketchpad::SetTextAlign (TAlign\_horizontal *tah* = LEFT, TAlign\_vertical *tav* = TOP) [inline, virtual]**

Set horizontal and vertical text alignment.

**Parameters:**

*tah* horizontal alignment

*tav* vertical alignment

**Default action:**

None.

**16.48.4.5 virtual DWORD oapi::Sketchpad::SetTextColor (DWORD *col*) [inline, virtual]**

Set the foreground colour for text output.

**Parameters:**

*col* colour description (format: 0xBBGGRR)

**Returns:**

Previous colour setting.

**Default action:**

None, returns 0.

**16.48.4.6 virtual DWORD oapi::Sketchpad::SetBackgroundColor (DWORD *col*) [inline, virtual]**

Set the background colour for text output.

**Parameters:**

*col* background colour description (format: 0xBBGGRR)

**Returns:**

Previous colour setting

**Default action:**

None, returns 0.

**Note:**

The background colour is only used if the background mode is set to BK\_OPAQUE.

**See also:**

[SetBackgroundMode](#)

**16.48.4.7 virtual void oapi::Sketchpad::SetBackgroundMode (BkgMode *mode*) [inline, virtual]**

Set the background mode for text output.

**Parameters:**

*mode* background mode (see [BkgMode](#))

**Default action:**

None.

**Note:**

In opaque background mode, the text background is drawn in the current background colour (see [SetBackgroundColor](#)).

The default background mode (before the first call of [SetBackgroundMode](#)) should be transparent.

**See also:**

[SetBackgroundColor](#), [SetTextColor](#)

**16.48.4.8 virtual DWORD oapi::Sketchpad::GetCharSize () [inline, virtual]**

Return height and (average) width of a character in the currently selected font.

**Returns:**

Height of character cell [pixel] in the lower 16 bit of the return value, and (average) width of character cell [pixel] in the upper 16 bit.

**Default action:**

None, returns 0.

**Note:**

The height value should describe the height of the character cell (i.e. the smallest box circumscribing all characters in the font), but without any "internal leading", i.e. the gap between characters in two consecutive lines.

For proportional fonts, the width value should be an approximate average character width.

**16.48.4.9 virtual DWORD oapi::Sketchpad::GetTextWidth (const char \* str, int len = 0) [inline, virtual]**

Return the width of a text string in the currently selected font.

**Parameters:**

*str* text string

*len* string length, or 0 for auto (0-terminated string)

**Returns:**

width of the string, drawn in the currently selected font [pixel]

**Default action:**

None, returns 0.

**See also:**

[SetFont](#)

**16.48.4.10 virtual void oapi::Sketchpad::SetOrigin (int x, int y) [inline, virtual]**

Set the position in the surface bitmap which is mapped to the origin of the coordinate system for all drawing functions.

**Parameters:**

*x* horizontal position of the origin [pixel]

*y* vertical position of the origin [pixel]

**Default action:**

None.

**Note:**

By default, the reference point for drawing function coordinates is the top left corner of the bitmap, with positive x-axis to the right, and positive y-axis down.

`SetOrigin` can be used to shift the logical reference point to a different position in the surface bitmap (but not to change the orientation of the axes).

If the drawing system used by an implementation does not support this function directly, the derived class should itself account for the shift in origin, by subtracting the offset from all coordinate values.

**16.48.4.11 virtual bool oapi::Sketchpad::Text (int *x*, int *y*, const char \* *str*, int *len*) [inline, virtual]**

Draw a text string.

**Parameters:**

*x* reference x position [pixel]  
*y* reference y position [pixel]  
*str* text string  
*len* string length for output

**Returns:**

*true* on success, *false* on failure.

**Default action:**

None, returns false.

**16.48.4.12 virtual bool oapi::Sketchpad::TextBox (int *x1*, int *y1*, int *x2*, int *y2*, const char \* *str*, int *len*) [virtual]**

Draw a text string into a rectangle.

**Parameters:**

*x1* left edge [pixel]  
*y1* top edge [pixel]  
*x2* right edge [pixel]  
*y2* bottom edge [pixel]  
*str* text string  
*len* string length for output

**Returns:**

*true* on success, *false* on failure.

**Default action:**

Implementation via [Text](#) calls.

**Note:**

This method should write the text string into the specified rectangle, using the current font. Line breaks should automatically be applied as required to fit the text in the box.

The bottom edge (*y2*) should probably be ignored, so text isn't truncated if it doesn't fit the box.

**16.48.4.13 virtual void oapi::Sketchpad::Pixel (int *x*, int *y*, DWORD *col*) [inline, virtual]**

Draw a single pixel in a specified colour.

**Parameters:**

- x* x-coordinate of point [pixel]
- y* y-coordinate of point [pixel]
- col* pixel colour (format: 0xBBGGRR)

**16.48.4.14 virtual void oapi::Sketchpad::MoveTo (int *x*, int *y*) [inline, virtual]**

Move the drawing reference to a new point.

**Parameters:**

- x* x-coordinate of new reference point [pixel]
- y* y-coordinate of new reference point [pixel]

**Note:**

Some methods use the drawing reference point for drawing operations, e.g. [LineTo](#).

**Default action:**

None.

**See also:**

[LineTo](#)

**16.48.4.15 virtual void oapi::Sketchpad::LineTo (int *x*, int *y*) [inline, virtual]**

Draw a line to a specified point.

**Parameters:**

- x* x-coordinate of line end point [pixel]
- y* y-coordinate of line end point [pixel]

**Default action:**

None.

**Note:**

The line starts at the current drawing reference point.

**See also:**

[MoveTo](#)

**16.48.4.16 virtual void oapi::Sketchpad::Line (int *x0*, int *y0*, int *x1*, int *y1*) [inline, virtual]**

Draw a line between two points.

**Parameters:**

*x0* x-coordinate of first point [pixel]  
*y0* y-coordinate of first point [pixel]  
*x1* x-coordinate of second point [pixel]  
*y1* y-coordinate of second point [pixel]

**Default action:**

None.

**Note:**

The line is drawn with the currently selected pen.

**See also:**

[SetPen](#)

**16.48.4.17 virtual void oapi::Sketchpad::Rectangle (int *x0*, int *y0*, int *x1*, int *y1*) [virtual]**

Draw a rectangle (filled or outline).

**Parameters:**

*x0* left edge of rectangle [pixel]  
*y0* top edge of rectangle [pixel]  
*x1* right edge of rectangle [pixel]  
*y1* bottom edge of rectangle [pixel]

**Default action:**

Draws the rectangle from 4 line segments by calling [MoveTo](#) and [LineTo](#).

**Note:**

Derived classes should overload this method if possible, because the default method does not allow to draw filled rectangles, and may be less efficient than a dedicated implementation.  
Implementations should fill the rectangle with the currently selected brush resource.

**See also:**

[MoveTo](#), [LineTo](#), [Ellipse](#), [Polygon](#)

**16.48.4.18 virtual void oapi::Sketchpad::Ellipse (int *x0*, int *y0*, int *x1*, int *y1*) [inline, virtual]**

Draw an ellipse from its bounding box.

**Parameters:**

*x0* left edge of bounding box [pixel]  
*y0* top edge of bounding box [pixel]  
*x1* right edge of bounding box [pixel]  
*y1* bottom edge of bounding box [pixel]

**Default action:**

None.

**Note:**

Implementations should fill the ellipse with the currently selected brush resource.

**See also:**

[Rectangle](#), [Polygon](#)

**16.48.4.19 virtual void oapi::Sketchpad::Polygon (const IVECTOR2 \* *pt*, int *npt*) [inline, virtual]**

Draw a closed polygon given by vertex points.

**Parameters:**

*pt* list of vertex points  
*npt* number of points in the list

**Default action:**

None.

**Note:**

Implementations should draw the outline of the polygon with the current pen, and fill it with the current brush.  
The polygon should be closed, i.e. the last point joined with the first one.

**See also:**

[Polyline](#), [PolyPolygon](#), [Rectangle](#), [Ellipse](#)

**16.48.4.20 virtual void oapi::Sketchpad::Polyline (const IVECTOR2 \* *pt*, int *npt*) [inline, virtual]**

Draw a line of piecewise straight segments.

**Parameters:**

*pt* list of vertex points  
*npt* number of points in the list

**Default action:**

None

**Note:**

Implementations should draw the line with the currently selected pen.

Polylines are open figures: the end points are not connected, and no fill operation is performed.

**See also:**

[Polygon](#), [PolyPolyline](#), [Rectangle](#), [Ellipse](#)

**16.48.4.21 virtual void oapi::Sketchpad::PolyPolygon (const IVECTOR2 \*pt, const int \*npt, const int nline) [virtual]**

Draw a set of polygons.

**Parameters:**

*pt* list of vertex points for all polygons

*npt* list of number of points for each polygon

*nline* number of polygons

**Default action:**

Calls [Polygon](#) for each line in the list.

**Note:**

The number of entries in *npt* must be  $\geq$  *nline*, and the number of points in *pt* must be at least the sum of the values in *npt*.

Implementations should overload this function if they can provide efficient direct support for it. Otherwise, the base class implementation should be sufficient.

**See also:**

[Polygon](#), [Polyline](#), [PolyPolyline](#)

**16.48.4.22 virtual void oapi::Sketchpad::PolyPolyline (const IVECTOR2 \*pt, const int \*npt, const int nline) [virtual]**

Draw a set of polylines.

**Parameters:**

*pt* list of vertex points for all lines

*npt* list of number of points for each line

*nline* number of lines

**Default action:**

Calls [Polyline](#) for each line in the list.

**Note:**

The number of entries in *npt* must be  $\geq$  *nline*, and the number of points in *pt* must be at least the sum of the values in *npt*.

Implementations should overload this function if they can provide efficient direct support for it. Otherwise, the base class implementation should be sufficient.

**See also:**

[Polyline](#), [Polygon](#), [PolyPolygon](#)

**16.48.4.23 SURFHANDLE oapi::Sketchpad::GetSurface () const [inline]**

Returns the surface associated with the drawing object.

**Returns:**

Surface handle

**16.48.4.24 virtual HDC oapi::Sketchpad::GetDC () [inline, virtual]**

Return the Windows device context handle, if applicable.

**Returns:**

device context handle

**Default action:**

None, returns NULL.

**Note:**

`Sketchpad` implementations based on the Windows GDI system should overload this function to return the device context handle here. All other implementations should not overload this function.

The device context returned by this function should not be released (e.g. with `ReleaseDC`). The device context is released automatically when the `Sketchpad` instance is destroyed.

This method should be regarded as temporary. Ultimately, the device-dependent drawing mechanism should be hidden outside the sketchpad implementation.

The documentation for this class was generated from the following file:

- Orbitersdk/include/DrawAPI.h

## 16.49 SpotLight Class Reference

```
#include <OrbiterAPI.h>
```

Inheritance diagram for SpotLight:

Collaboration diagram for SpotLight:

#### 16.49.1 Detailed Description

Class for directed spot light sources.

#### Public Member Functions

- `SpotLight (OBJHANDLE hObj, const VECTOR3 &_pos, const VECTOR3 &_dir, double _range, double att0, double att1, double att2, double _umbra, double _penumbra)`  
*Creates a white spotlight.*
- `SpotLight (OBJHANDLE hObj, const VECTOR3 &_pos, const VECTOR3 &_dir, double _range, double att0, double att1, double att2, double _umbra, double _penumbra, COLOUR4 diffuse, COLOUR4 specular, COLOUR4 ambient)`  
*Creates a coloured spotlight.*
- `double GetUmbra () const`  
*Returns the angular aperture of inner (maximum intensity) cone.*
- `double GetPenumbra () const`  
*Returns the angular aperture of outer (zero intensity) cone.*
- `void SetAperture (double _umbra, double _penumbra)`  
*Set the spotlight cone geometry.*

#### Protected Attributes

- `double umbra`
- `double penumbra`

### 16.49.2 Constructor & Destructor Documentation

#### 16.49.2.1 `SpotLight::SpotLight (OBJHANDLE hObj, const VECTOR3 & pos, const VECTOR3 & dir, double range, double att0, double att1, double att2, double umbra, double penumbra)`

Creates a white spotlight.

##### Parameters:

- hObj*** handle of object the spotlight is attached to
- pos*** spotlight position in local object coordinates [m]
- dir*** spotlight direction in local object coordinates
- range*** spotlight range [m]
- att0*** light attenuation parameters
- att1*** light attenuation parameters
- att2*** light attenuation parameters
- umbra*** angular aperture of inner (maximum intensity) cone [rad]
- penumbra*** angular aperture of outer (zero intensity) cone [rad]

##### Note:

Direction vector *dir* must be normalised to length 1.

$0 < \text{umbra} \leq \text{penumbra} \leq \pi$  is required.

The intensity falloff between *umbra* and *penumbra* is linear from maximum intensity to zero.

#### 16.49.2.2 `SpotLight::SpotLight (OBJHANDLE hObj, const VECTOR3 & pos, const VECTOR3 & dir, double range, double att0, double att1, double att2, double umbra, double penumbra, COLOUR4 diffuse, COLOUR4 specular, COLOUR4 ambient)`

Creates a coloured spotlight.

##### Parameters:

- hObj*** handle of object the spotlight is attached to
- pos*** spotlight position in local object coordinates [m]
- dir*** spotlight direction in local object coordinates
- range*** spotlight range [m]
- att0*** light attenuation parameters
- att1*** light attenuation parameters
- att2*** light attenuation parameters
- umbra*** angular aperture of inner (maximum intensity) cone [rad]
- penumbra*** angular aperture of outer (zero intensity) cone [rad]
- diffuse*** light source's contribution to lit objects' diffuse colour component
- specular*** light source's contribution to lit objects' specular colour component
- ambient*** light source's contribution to lit objects' ambient colour component

##### Note:

Direction vector *dir* must be normalised to length 1.

$0 < \text{umbra} \leq \text{penumbra} \leq \pi$  is required.

The intensity falloff between *umbra* and *penumbra* is linear from maximum intensity to zero.

### 16.49.3 Member Function Documentation

#### 16.49.3.1 double SpotLight::GetUmbra () const [inline]

Returns the angular aperture of inner (maximum intensity) cone.

##### Returns:

Aperture of inner spotlight cone [rad]

##### See also:

[GetPenumbra](#)

#### 16.49.3.2 double SpotLight::GetPenumbra () const [inline]

Returns the angular aperture of outer (zero intensity) cone.

##### Returns:

Aperture of outer spotlight cone [rad]

##### See also:

[GetUmbra](#)

#### 16.49.3.3 void SpotLight::SetAperture (double *\_umbra*, double *\_penumbra*)

Set the spotlight cone geometry.

##### Parameters:

*\_umbra* angular aperture of inner (maximum intensity) cone [rad]

*\_penumbra* angular aperture of outer (zero intensity) cone [rad]

The documentation for this class was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 16.50 VECTOR3 Union Reference

```
#include <OrbiterAPI.h>
```

### 16.50.1 Detailed Description

3-element vector

#### Public Attributes

- double *data* [3]  
*array data interface*

```
• struct {
    double x
    double y
    double z
};
```

*named data interface*

The documentation for this union was generated from the following file:

- OrbiterSDK/include/OrbiterAPI.h

## 16.51 VESSEL Class Reference

```
#include <VesselAPI.h>
```

Inheritance diagram for VESSEL:

### 16.51.1 Detailed Description

Base class for objects of vessel type (spacecraft and similar).

VESSEL is the base class for addon modules of 'vessel' type (spacecraft, space stations, satellites, deep space probes, etc.) This class defines the interface between the module's vessel definition and the parameters maintained internally by Orbiter to define the vessel state. It provides access to the various status parameters and methods of individual spacecraft.

It is important to note that a VESSEL instance represents an *interface* to an existing vessel in Orbiter, rather than the vessel itself. Vessels can exist without a corresponding VESSEL instance, and deleting a VESSEL instance does not delete the vessel.

Most of the methods provided by the VESSEL class are of 'get' and 'set' type, i.e. for retrieving vessel parameter states, or modifying them. It does *not* define any callback functions that Orbiter uses to notify the vessel of events. These are implemented in the [VESSEL2](#) class (derived from VESSEL). The latest version of the interface is [VESSEL3](#), which implements additional functions. User-defined vessel classes should therefore be derived from VESSEL3 instead of VESSEL.

For complete vessel module implementations, see the examples in OrbiterSDK\samples, for example OrbiterSDK\samples\ShuttlePB.

### Construction/creation, handles and interfaces

- **VESSEL** (**OBJHANDLE** hVessel, int fmodel=1)  
*Creates a VESSEL interface instance from a vessel handle.*
- int **Version** () const  
*Returns the version number of the vessel interface class.*
- const **OBJHANDLE** **GetHandle** () const  
*Returns a handle to the vessel object.*
- bool **GetEditorModule** (char \*fname) const  
*Returns the file name of the DLL containing the vessel's scenario editor extensions.*

### General vessel properties

- char \* **GetName** () const  
*Returns the vessel's name.*
- char \* **GetClassName** () const  
*Returns the vessel's class name.*
- int **GetFlightModel** () const  
*Returns the requested realism level for the flight model.*
- int **GetDamageModel** () const  
*Returns the current user setting for damage and systems failure simulation.*
- bool **GetEnableFocus** () const  
*Returns true if the vessel can receive the input focus, false otherwise.*
- void **SetEnableFocus** (bool enable) const  
*Enable or disable the vessel's ability to receive the input focus.*
- double **GetSize** () const  
*Returns the vessel's mean radius.*
- void **SetSize** (double size) const  
*Set the vessel's mean radius.*
- void **SetVisibilityLimit** (double vislimit, double spotlimit=-1) const  
*Defines the vessel's range of visibility.*
- double **GetClipRadius** () const  
*Returns the radius of the vessel's circumscribing sphere.*
- void **SetAlbedoRGB** (const **VECTOR3** &albedo) const  
*Set the average colour distribution reflected by the vessel.*

- void [SetClipRadius](#) (double rad) const  
*Set the radius of the vessel's circumscribing sphere.*
- double [GetEmptyMass](#) () const  
*Returns the vessel's empty mass (excluding propellants).*
- void [SetEmptyMass](#) (double m) const  
*Set the vessel's empty mass (excluding propellants).*
- double [GetCOG\\_elev](#) () const  
*Elevation of the vessel's centre of gravity (COG) above ground.*
- void [GetTouchdownPoints](#) (VECTOR3 &pt1, VECTOR3 &pt2, VECTOR3 &pt3) const  
*Returns the three points defining the vessel's ground contact plane.*
- void [SetTouchdownPoints](#) (const VECTOR3 &pt1, const VECTOR3 &pt2, const VECTOR3 &pt3) const  
*Defines the three points defining the vessel's ground contact plane.*
- void [SetSurfaceFrictionCoeff](#) (double mu\_lng, double mu\_lat) const  
*Set friction coefficients for ground contact.*
- void [GetCrossSections](#) (VECTOR3 &cs) const  
*Returns the vessel's cross sections projected in the direction of the vessel's principal axes.*
- void [SetCrossSections](#) (const VECTOR3 &cs) const  
*Defines the vessel's cross-sectional areas, projected in the directions of the vessel's principal axes.*
- void [GetPMI](#) (VECTOR3 &pmi) const  
*Returns the vessel's mass-normalised principal moments of inertia (PMI).*
- void [SetPMI](#) (const VECTOR3 &pmi) const  
*Set the vessel's mass-normalised principal moments of inertia (PMI).*
- double [GetGravityGradientDamping](#) () const  
*Returns the vessel's damping coefficient for gravity field gradient-induced torque.*
- bool [SetGravityGradientDamping](#) (double damp) const  
*Sets the vessel's damping coefficient for gravity field gradient-induced torque.*

### Vessel state

- void [GetStatus](#) (VESSELSTATUS &status) const  
*Returns the vessel's current status parameters in a VESSELSTATUS structure.*
- void [GetStatusEx](#) (void \*status) const  
*Returns the vessel's current status parameters in a VESSELSTATUSx structure (version x >= 2).*
- void [DefSetState](#) (const VESSELSTATUS \*status) const

*Set default vessel status parameters.*

- void **DefSetStateEx** (const void \*status) const  
*Set default vessel status parameters.*
- DWORD **GetFlightStatus** () const  
*Returns a bit flag defining the vessel's current flight status.*
- double **GetMass** () const  
*Returns current (total) vessel mass.*
- void **GetGlobalPos** (VECTOR3 &pos) const  
*Returns the vessel's current position in the global reference frame.*
- void **GetGlobalVel** (VECTOR3 &vel) const  
*Returns the vessel's current velocity in the global reference frame.*
- void **GetRelativePos** (OBJHANDLE hRef, VECTOR3 &pos) const  
*Returns the vessel's current position with respect to another object.*
- void **GetRelativeVel** (OBJHANDLE hRef, VECTOR3 &vel) const  
*Returns the vessel's current velocity relative to another object.*
- void **GetAngularVel** (VECTOR3 &avel) const  
*Returns the vessel's current angular velocity components around its principal axes.*
- void **GetAngularAcc** (VECTOR3 &aacc) const  
*Returns the vessel's current angular acceleration components around its principal axes.*
- void **GetLinearMoment** (VECTOR3 &F) const  
*Returns the linear force vector currently acting on the vessel.*
- void **GetAngularMoment** (VECTOR3 &amom) const  
*Returns the sum of angular moments currently acting on the vessel.*
- void **SetAngularVel** (const VECTOR3 &avel) const  
*Applies new angular velocity to the vessel.*
- void **GetGlobalOrientation** (VECTOR3 &arot) const  
*Returns the Euler angles defining the vessel's orientation.*
- void **SetGlobalOrientation** (const VECTOR3 &arot) const  
*Sets the vessel's orientation via Euler angles.*
- bool **GroundContact** () const  
*Returns a flag indicating contact with a planetary surface.*
- bool **OrbitStabilised** () const  
*Flag indicating whether orbit stabilisation is used for the vessel at the current time step.*

- bool [NonsphericalGravityEnabled \(\) const](#)  
*Flag for nonspherical gravity perturbations.*
- DWORD [GetADCtrlMode \(\) const](#)  
*Returns aerodynamic control surfaces currently under manual control.*
- void [SetADCtrlMode \(DWORD mode\) const](#)  
*Configure manual input mode for aerodynamic control surfaces.*
- bool [ActivateNavmode \(int mode\)](#)  
*Activates one of the automated orbital navigation modes.*
- bool [DeactivateNavmode \(int mode\)](#)  
*Deactivates an automated orbital navigation mode.*
- bool [ToggleNavmode \(int mode\)](#)  
*Toggles a navigation mode on/off.*
- bool [GetNavmodeState \(int mode\)](#)  
*Returns the current active/inactive state of a navigation mode.*

## Orbital elements

See also: [Basics of orbital mechanics](#)

- const [OBJHANDLE GetGravityRef \(\) const](#)  
*Returns a handle to the main contributor of the gravity field at the vessel's current position.*
- [OBJHANDLE GetElements \(ELEMENTS &el, double &mjd\\_ref\) const](#)  
*Returns osculating orbital elements.*
- bool [GetElements \(OBJHANDLE hRef, ELEMENTS &el, ORBITPARAM \\*prm=0, double mjd\\_ref=0, int frame=FRAME\\_ECL\) const](#)  
*Returns osculating elements and additional orbit parameters.*
- bool [SetElements \(OBJHANDLE hRef, const ELEMENTS &el, ORBITPARAM \\*prm=0, double mjd\\_ref=0, int frame=FRAME\\_ECL\) const](#)  
*Set vessel state (position and velocity) by means of a set of osculating orbital elements.*
- [OBJHANDLE GetSMi \(double &smi\) const](#)  
*Returns the magnitude of the semi-minor axis of the current osculating orbit.*
- [OBJHANDLE GetArgPer \(double &arg\) const](#)  
*Returns argument of periapsis of the current osculating orbit.*
- [OBJHANDLE GetPeDist \(double &pedist\) const](#)  
*Returns the periapsis distance of the current osculating orbit.*
- [OBJHANDLE GetApDist \(double &apdist\) const](#)  
*Returns the apoapsis distance of the current osculating orbit.*

### Surface-relative parameters

- const **OBJHANDLE GetSurfaceRef () const**  
*Returns a handle to the surface reference object (planet or moon).*
- double **GetAltitude () const**  
*Returns the altitude above the surface of the surface reference body.*
- double **GetPitch () const**  
*Returns the current pitch angle with respect to the local horizon.*
- double **GetBank () const**  
*Returns the current bank (roll) angle with respect to the local horizon.*
- double **GetYaw () const**  
*Returns the current yaw angle with respect to the local horizon.*
- **OBJHANDLE GetEquPos (double &longitude, double &latitude, double &radius) const**  
*Returns vessel's current equatorial position with respect to the closest planet or moon.*

### Atmospheric parameters

- const **OBJHANDLE GetAtmRef () const**  
*Returns a handle to the reference body for atmospheric calculations.*
- double **GetAtmTemperature () const**  
*Returns ambient atmospheric temperature at current vessel position.*
- double **GetAtmDensity () const**  
*Returns atmospheric density at current vessel position.*
- double **GetAtmPressure () const**  
*Returns static atmospheric pressure at current vessel position.*

### Aerodynamic state parameters

- double **GetDynPressure () const**  
*Returns the current dynamic pressure for the vessel.*
- double **GetMachNumber () const**  
*Returns the vessel's current Mach number.*
- double **GetGroundspeed () const**  
*Returns magnitude of the ground speed vector.*
- bool **GetGroundspeedVector (REFFFRAME frame, VECTOR3 &v) const**  
*Returns the vessel's ground speed vector.*

- double [GetAirspeed](#) () const  
*Returns magnitude of the true airspeed vector.*
- bool [GetAirspeedVector](#) (REFFRAME frame, VECTOR3 &v) const  
*Returns the vessel's true airspeed vector.*
- bool [GetHorizonAirspeedVector](#) (VECTOR3 &v) const  
*Returns the airspeed vector in local horizon coordinates.*
- bool [GetShipAirspeedVector](#) (VECTOR3 &v) const  
*Returns the airspeed vector in vessel coordinates.*
- double [GetAOA](#) () const  
*Returns the current angle of attack.*
- double [GetSlipAngle](#) () const  
*Returns the lateral (yaw) angle between the velocity vector and the vessel's longitudinal axis.*

### Airfoils and control surfaces

- void [CreateAirfoil](#) (AIRFOIL\_ORIENTATION align, const VECTOR3 &ref, AirfoilCoeffFunc cf, double c, double S, double A) const  
*Creates a new airfoil and defines its aerodynamic properties.*
- AIRFOILHANDLE [CreateAirfoil2](#) (AIRFOIL\_ORIENTATION align, const VECTOR3 &ref, AirfoilCoeffFunc cf, double c, double S, double A) const  
*Creates a new airfoil and defines its aerodynamic properties.*
- AIRFOILHANDLE [CreateAirfoil3](#) (AIRFOIL\_ORIENTATION align, const VECTOR3 &ref, AirfoilCoeffFuncEx cf, void \*context, double c, double S, double A) const  
*Creates a new airfoil and defines its aerodynamic properties.*
- bool [GetAirfoilParam](#) (AIRFOILHANDLE hAirfoil, VECTOR3 \*ref, AirfoilCoeffFunc \*cf, void \*\*context, double \*c, double \*S, double \*A) const  
*Returns the parameters of an existing airfoil.*
- void [EditAirfoil](#) (AIRFOILHANDLE hAirfoil, DWORD flag, const VECTOR3 &ref, AirfoilCoeffFunc cf, double c, double S, double A) const  
*Resets the parameters of an existing airfoil definition.*
- bool [DelAirfoil](#) (AIRFOILHANDLE hAirfoil) const  
*Deletes a previously defined airfoil.*
- void [ClearAirfoilDefinitions](#) () const  
*Removes all airfoils currently defined for the vessel.*
- void [CreateControlSurface](#) (AIRCTRL\_TYPE type, double area, double dCl, const VECTOR3 &ref, int axis=AIRCTRL\_AXIS\_AUTO, UINT anim=(UINT)-1) const  
*Creates an aerodynamic control surface.*

- `CTRLSURFHANDLE CreateControlSurface2 (AIRCTRL_TYPE type, double area, double dCl, const VECTOR3 &ref, int axis=AIRCTRL_AXIS_AUTO, UINT anim=(UINT)-1) const`  
*Creates an aerodynamic control surface and returns a handle.*
- `CTRLSURFHANDLE CreateControlSurface3 (AIRCTRL_TYPE type, double area, double dCl, const VECTOR3 &ref, int axis=AIRCTRL_AXIS_AUTO, double delay=1.0, UINT anim=(UINT)-1) const`  
*Creates an aerodynamic control surface and returns a handle.*
- `bool DelControlSurface (CTRLSURFHANDLE hCtrlSurf) const`  
*Deletes a previously defined aerodynamic control surface.*
- `void ClearControlSurfaceDefinitions () const`  
*Removes all aerodynamic control surfaces.*
- `void SetControlSurfaceLevel (AIRCTRL_TYPE type, double level) const`  
*Updates the position of an aerodynamic control surface.*
- `void SetControlSurfaceLevel (AIRCTRL_TYPE type, double level, bool direct) const`  
*Updates the position of an aerodynamic control surface.*
- `double GetControlSurfaceLevel (AIRCTRL_TYPE type) const`  
*Returns the current position of a control surface.*
- `void CreateVariableDragElement (double *drag, double factor, const VECTOR3 &ref) const`  
*Attaches a modifiable drag component to the vessel.*
- `void ClearVariableDragElements () const`  
*Removes all drag elements defined with CreateVariableDragElement.*

### Aerodynamic vessel properties (legacy model)

The methods in this group are used only if the vessel does not define any airfoils.

- `void GetCW (double &cw_z_pos, double &cw_z_neg, double &cw_x, double &cw_y) const`  
*Returns the vessel's wind resistance coefficients (legacy flight model only).*
- `void SetCW (double cw_z_pos, double cw_z_neg, double cw_x, double cw_y) const`  
*Set the vessel's wind resistance coefficients along its axis directions.*
- `double GetWingAspect () const`  
*Returns the vessel's wing aspect ratio ( $wingspan^2 / wing\ area$ ).*
- `void SetWingAspect (double aspect) const`  
*Set the wing aspect ratio ( $wingspan^2 / wing\ area$ ).*
- `double GetWingEffectiveness () const`

*Returns the wing form factor used in aerodynamic calculations.*

- void **SetWingEffectiveness** (double eff) const  
*Set the wing form factor for aerodynamic lift and drag calculations.*
- void **GetRotDrag** (VECTOR3 &rd) const  
*Returns the vessel's atmospheric rotation resistance coefficients.*
- void **SetRotDrag** (const VECTOR3 &rd) const  
*Set the vessel's atmospheric rotation resistance coefficients.*
- double **GetPitchMomentScale** () const  
*Returns the scaling factor for the pitch moment.*
- void **SetPitchMomentScale** (double scale) const  
*Sets the scaling factor for the pitch moment.*
- double **GetYawMomentScale** () const  
*Returns the scaling factor for the yaw moment.*
- void **SetYawMomentScale** (double scale) const  
*Sets the scaling factor for the yaw moment.*
- double **GetTrimScale** () const  
*Returns the scaling factor for the pitch trim control.*
- void **SetTrimScale** (double scale) const  
*Sets the scaling factor for the pitch trim control.*
- void **SetLiftCoeffFunc** (LiftCoeffFunc lcf) const  
*Defines the callback function for aerodynamic lift calculation.*

## Forces

- double **GetLift** () const  
*Returns magnitude of aerodynamic lift force vector.*
- double **GetDrag** () const  
*Returns magnitude of aerodynamic drag force vector.*
- bool **GetWeightVector** (VECTOR3 &G) const  
*Returns gravitational force vector in local vessel coordinates.*
- bool **GetThrustVector** (VECTOR3 &T) const  
*Returns thrust force vector in local vessel coordinates.*
- bool **GetLiftVector** (VECTOR3 &L) const  
*Returns aerodynamic lift force vector in local vessel coordinates.*

- bool [GetDragVector \(VECTOR3 &D\) const](#)  
*Returns aerodynamic drag force vector in local vessel coordinates.*
- bool [GetForceVector \(VECTOR3 &F\) const](#)  
*Returns total force vector acting on the vessel in local vessel coordinates.*
- bool [GetTorqueVector \(VECTOR3 &M\) const](#)  
*Returns the total torque vector acting on the vessel in local vessel coordinates.*
- void [AddForce \(const VECTOR3 &F, const VECTOR3 &r\) const](#)  
*Add a custom body force.*

## Fuel management

- PROPELLANT\_HANDLE [CreatePropellantResource \(double maxmass, double mass=-1.0, double efficiency=1.0\) const](#)  
*Create a new propellant resource ("fuel tank").*
- void [DelPropellantResource \(PROPELLANT\\_HANDLE &ph\) const](#)  
*Remove a propellant resource.*
- void [ClearPropellantResources \(\) const](#)  
*Remove all propellant resources for the vessel.*
- DWORD [GetPropellantCount \(\) const](#)  
*Returns the current number of vessel propellant resources.*
- PROPELLANT\_HANDLE [GetPropellantHandleByIndex \(DWORD idx\) const](#)  
*Returns the handle of a propellant resource for a given index.*
- double [GetPropellantMaxMass \(PROPELLANT\\_HANDLE ph\) const](#)  
*Returns the maximum capacity of a propellant resource.*
- void [SetPropellantMaxMass \(PROPELLANT\\_HANDLE ph, double maxmass\) const](#)  
*Reset the maximum capacity of a fuel resource.*
- double [GetPropellantMass \(PROPELLANT\\_HANDLE ph\) const](#)  
*Returns the current mass of a propellant resource.*
- void [SetPropellantMass \(PROPELLANT\\_HANDLE ph, double mass\) const](#)  
*Reset the current mass of a propellant resource.*
- double [GetTotalPropellantMass \(\) const](#)  
*Returns the vessel's current total propellant mass.*
- double [GetPropellantEfficiency \(PROPELLANT\\_HANDLE ph\) const](#)  
*Returns the efficiency factor of a propellant resource.*
- void [SetPropellantEfficiency \(PROPELLANT\\_HANDLE ph, double efficiency\) const](#)

*Reset the efficiency factor of a fuel resource.*

- double **GetPropellantFlowrate** (**PROPELLANT\_HANDLE** ph) const  
*Returns the current mass flow rate from a propellant resource.*
- double **GetTotalPropellantFlowrate** () const  
*Returns the current total mass flow rate, summed over all propellant resources.*
- void **SetDefaultPropellantResource** (**PROPELLANT\_HANDLE** ph) const  
*Define a "default" propellant resource.*
- **PROPELLANT\_HANDLE** **GetDefaultPropellantResource** () const  
*Returns the handle for the vessel's default propellant resource.*
- double **GetMaxFuelMass** () const  
*Returns the maximum capacity of the vessel's default propellant resource.*
- void **SetMaxFuelMass** (double mass) const  
*Set the maximum fuel capacity of the vessel's default propellant resource.*
- double **GetFuelMass** () const  
*Returns the current mass of the vessel's default propellant resource.*
- void **SetFuelMass** (double mass) const  
*Reset the current mass of the vessel's default propellant resource.*
- double **GetFuelRate** () const  
*Returns the current mass flow rate from the default propellant resource.*

### Thruster management

- **THRUSTER\_HANDLE** **CreateThruster** (const **VECTOR3** &pos, const **VECTOR3** &dir, double maxth0, **PROPELLANT\_HANDLE** hp=NULL, double isp0=0.0, double isp\_ref=0.0, double p\_ref=101.4e3) const  
*Add a logical thruster definition for the vessel.*
- bool **DelThruster** (**THRUSTER\_HANDLE** &th) const  
*Delete a logical thruster definition.*
- void **ClearThrusterDefinitions** () const  
*Delete all thruster and thruster group definitions.*
- DWORD **GetThrusterCount** () const  
*Returns the number of thrusters currently defined.*
- **THRUSTER\_HANDLE** **GetThrusterHandleByIndex** (DWORD idx) const  
*Returns the handle of a thruster specified by its index.*
- **PROPELLANT\_HANDLE** **GetThrusterResource** (**THRUSTER\_HANDLE** th) const

*Returns a handle for the propellant resource feeding the thruster.*

- void `SetThrusterResource (THRUSTER_HANDLE th, PROPELLANT_HANDLE ph) const`  
*Connect a thruster to a propellant resource.*
- void `GetThrusterRef (THRUSTER_HANDLE th, VECTOR3 &pos) const`  
*Returns the thrust force attack point of a thruster.*
- void `SetThrusterRef (THRUSTER_HANDLE th, const VECTOR3 &pos) const`  
*Reset the thrust force attack point of a thruster.*
- void `GetThrusterDir (THRUSTER_HANDLE th, VECTOR3 &dir) const`  
*Returns the force direction of a thruster.*
- void `SetThrusterDir (THRUSTER_HANDLE th, const VECTOR3 &dir) const`  
*Reset the force direction of a thruster.*
- double `GetThrusterMax0 (THRUSTER_HANDLE th) const`  
*Returns the maximum vacuum thrust rating of a thruster.*
- void `SetThrusterMax0 (THRUSTER_HANDLE th, double maxth0) const`  
*Reset the maximum vacuum thrust rating of a thruster.*
- double `GetThrusterMax (THRUSTER_HANDLE th) const`  
*Returns the current maximum thrust rating of a thruster.*
- double `GetThrusterMax (THRUSTER_HANDLE th, double p_ref) const`  
*Returns the maximum thrust rating of a thruster at a specific ambient pressure.*
- double `GetThrusterIsp0 (THRUSTER_HANDLE th) const`  
*Returns the vacuum fuel-specific impulse (Isp) rating for a thruster.*
- double `GetThrusterIsp (THRUSTER_HANDLE th) const`  
*Returns the current fuel-specific impulse (Isp) rating of a thruster.*
- double `GetThrusterIsp (THRUSTER_HANDLE th, double p_ref) const`  
*Returns the fuel-specific impulse (Isp) rating of a thruster at a specific ambient atmospheric pressure.*
- void `SetThrusterIsp (THRUSTER_HANDLE th, double isp) const`  
*Reset the fuel-specific impulse (Isp) rating of a thruster, assuming no pressure dependence.*
- void `SetThrusterIsp (THRUSTER_HANDLE th, double isp0, double isp_ref, double p_ref=101.4e3) const`  
*Reset the fuel-specific impulse (Isp) rating of a thruster including a pressure dependency.*
- double `GetThrusterLevel (THRUSTER_HANDLE th) const`  
*Returns the current thrust level setting of a thruster.*
- void `SetThrusterLevel (THRUSTER_HANDLE th, double level) const`  
*Set thrust level for a thruster.*

- void **IncThrusterLevel** (**THRUSTER\_HANDLE** th, double dlevel) const  
*Apply a change to the thrust level of a thruster.*
- void **SetThrusterLevel\_SingleStep** (**THRUSTER\_HANDLE** th, double level) const  
*Set the thrust level of a thruster for the current time step only.*
- void **IncThrusterLevel\_SingleStep** (**THRUSTER\_HANDLE** th, double dlevel) const  
*Apply a thrust level change to a thruster for the current time step only.*
- void **GetThrusterMoment** (**THRUSTER\_HANDLE** th, **VECTOR3** &F, **VECTOR3** &T) const  
*Returns the linear moment (force) and angular moment (torque) currently generated by a thruster.*
- double **GetISP** () const  
*Returns the vessel's current default fuel-specific impulse.*
- void **SetISP** (double isp) const  
*Sets the default Isp value for subsequently created thrusters.*

### Thruster group management

- **THGROUP\_HANDLE CreateThrusterGroup** (**THRUSTER\_HANDLE** \*th, int nth, **THGROUP\_TYPE** thgt) const  
*Combine thrusters into a logical group.*
- bool **DelThrusterGroup** (**THGROUP\_HANDLE** thg, bool delth=false) const  
*Delete a thruster group and (optionally) all associated thrusters.*
- bool **DelThrusterGroup** (**THGROUP\_TYPE** thgt, bool delth=false) const  
*Delete a default thruster group and (optionally) all associated thrusters.*
- **THGROUP\_HANDLE GetThrusterGroupHandle** (**THGROUP\_TYPE** thgt) const  
*Returns the handle of a default thruster group.*
- **THGROUP\_HANDLE GetUserThrusterGroupHandleByIndex** (DWORD idx) const  
*Returns the handle of a user-defined (nonstandard) thruster group.*
- DWORD **GetGroupThrusterCount** (**THGROUP\_HANDLE** thg) const  
*Returns the number of thrusters assigned to a logical thruster group.*
- DWORD **GetGroupThrusterCount** (**THGROUP\_TYPE** thgt) const  
*Returns the number of thrusters assigned to a standard logical thruster group.*
- **THRUSTER\_HANDLE GetGroupThruster** (**THGROUP\_HANDLE** thg, DWORD idx) const  
*Returns a handle for a thruster that belongs to a specified thruster group.*
- **THRUSTER\_HANDLE GetGroupThruster** (**THGROUP\_TYPE** thgt, DWORD idx) const  
*Returns a handle for a thruster that belongs to a standard thruster group.*

- **DWORD GetUserThrusterGroupCount () const**  
*Returns the number of user-defined (nonstandard) thruster groups.*
- **bool ThrusterGroupDefined (THGROUP\_TYPE thgt) const**  
*Indicates if a default thruster group is defined by the vessel.*
- **void SetThrusterGroupLevel (THGROUP\_HANDLE thg, double level) const**  
*Sets the thrust level for all thrusters in a group.*
- **void SetThrusterGroupLevel (THGROUP\_TYPE thgt, double level) const**  
*Sets the thrust level for all thrusters in a standard group.*
- **void IncThrusterGroupLevel (THGROUP\_HANDLE thg, double dlevel) const**  
*Increments the thrust level for all thrusters in a group.*
- **void IncThrusterGroupLevel (THGROUP\_TYPE thgt, double dlevel) const**  
*Increments the thrust level for all thrusters in a standard group.*
- **void IncThrusterGroupLevel\_SingleStep (THGROUP\_HANDLE thg, double dlevel) const**  
*Increments the thrust level of a group for a single time step.*
- **void IncThrusterGroupLevel\_SingleStep (THGROUP\_TYPE thgt, double dlevel) const**  
*Increments the thrust level of a standard group for a single time step.*
- **double GetThrusterGroupLevel (THGROUP\_HANDLE thg) const**  
*Returns the mean thrust level for a thruster group.*
- **double GetThrusterGroupLevel (THGROUP\_TYPE thgt) const**  
*Returns the mean thrust level for a default thruster group.*
- **double GetManualControlLevel (THGROUP\_TYPE thgt, DWORD mode=MANCTRL\_ATTMODE, DWORD device=MANCTRL\_ANYDEVICE) const**  
*Returns the thrust level of an attitude thruster group set via keyboard or mouse input.*

### Reaction control system

- **int GetAttitudeMode () const**  
*Returns the current RCS (reaction control system) thruster mode.*
- **bool SetAttitudeMode (int mode) const**  
*Sets the vessel's RCS (reaction control system) thruster mode.*
- **int ToggleAttitudeMode () const**  
*Switch between linear and rotational RCS mode.*
- **void GetAttitudeRotLevel (VECTOR3 &th) const**  
*Returns the current combined thrust levels for the reaction control system thruster groups in rotational mode.*

- void [SetAttitudeRotLevel](#) (const **VECTOR3** &th) const  
*Set RCS thruster levels for rotation in all 3 vessel axes.*
- void [SetAttitudeRotLevel](#) (int axis, double th) const  
*Set RCS thruster level for rotation around a single axis.*
- void [GetAttitudeLinLevel](#) (**VECTOR3** &th) const  
*Returns the current combined thrust levels for the reaction control system thruster groups in linear (translational) mode.*
- void [SetAttitudeLinLevel](#) (const **VECTOR3** &th) const  
*Set RCS thruster levels for linear translation in all 3 vessel axes.*
- void [SetAttitudeLinLevel](#) (int axis, double th) const  
*Set RCS thruster level for linear translation along a single axis.*

### Communication interface

- int [SendBufferedKey](#) (DWORD key, bool down=true, char \*kstate=0)  
*Send a simulated buffered key event to the vessel.*

### Navigation radio interface

- void [InitNavRadios](#) (DWORD nnav) const  
*Defines the number of navigation (NAV) radio receivers supported by the vessel.*
- DWORD [GetNavCount](#) () const  
*Returns the number of NAV radio receivers.*
- bool [SetNavChannel](#) (DWORD n, DWORD ch) const  
*Sets the channel of a NAV radio receiver.*
- DWORD [GetNavChannel](#) (DWORD n) const  
*Returns the current channel setting of a NAV radio receiver.*
- float [GetNavRecvFreq](#) (DWORD n) const  
*Returns the current radio frequency of a NAV radio receiver.*
- void [EnableTransponder](#) (bool enable) const  
*Enable/disable transmission of transponder signal.*
- bool [SetTransponderChannel](#) (DWORD ch) const  
*Switch the channel number of the vessel's transponder.*
- void [EnableIDS](#) (**DOCKHANDLE** hDock, bool bEnable) const  
*Enable/disable one of the vessel's IDS (Instrument Docking System) transmitters.*

- bool `SetIDSChannel (DOCKHANDLE hDock, DWORD ch) const`  
*Switch the channel number of one of the vessel's IDS (Instrument Docking System) transmitters.*
- `NAVHANDLE GetTransponder () const`  
*Return handle of vessel transponder if available.*
- `NAVHANDLE GetIDS (DOCKHANDLE hDock) const`  
*Return handle of one of the vessel's instrument docking system (IDS) radio transmitters.*
- `NAVHANDLE GetNavSource (DWORD n) const`  
*Return handle of transmitter source currently received by one of the vessel's NAV receivers.*

### Cockpit camera methods

- void `SetCameraOffset (const VECTOR3 &co) const`  
*Set the camera position for internal (cockpit) view.*
- void `GetCameraOffset (VECTOR3 &co) const`  
*Returns the current camera position for internal (cockpit) view.*
- void `SetCameraDefaultDirection (const VECTOR3 &cd) const`  
*Set the default camera direction for internal (cockpit) view.*
- void `SetCameraDefaultDirection (const VECTOR3 &cd, double tilt) const`  
*Set the default camera direction and tilt angle for internal (cockpit) view.*
- void `GetCameraDefaultDirection (VECTOR3 &cd) const`  
*Returns the default camera direction for internal (cockpit) view.*
- void `SetCameraCatchAngle (double cangle) const`  
*Set the angle over which the cockpit camera auto-centers to default direction.*
- void `SetCameraRotationRange (double left, double right, double up, double down) const`  
*Sets the range over which the cockpit camera can be rotated from its default direction.*
- void `SetCameraShiftRange (const VECTOR3 &fpos, const VECTOR3 &lpos, const VECTOR3 &rpos) const`  
*Set the linear movement range for the cockpit camera.*
- void `SetCameraMovement (const VECTOR3 &fpos, double fphi, double ftlt, const VECTOR3 &lpos, double lphi, double ltlt, const VECTOR3 &rpos, double rphi, double rtlt) const`  
*Set both linear movement range and orientation of the cockpit camera when "leaning" forward, left and right.*

### Mesh methods

- void [ClearMeshes](#) (bool retain\_anim) const  
*Remove all mesh definitions for the vessel.*
- UINT [AddMesh](#) (const char \*meshname, const [VECTOR3](#) \*ofs=0) const  
*Load a mesh definition for the vessel from a file.*
- UINT [AddMesh](#) ([MESHHANDLE](#) hMesh, const [VECTOR3](#) \*ofs=0) const  
*Add a pre-loaded mesh definition to the vessel.*
- UINT [InsertMesh](#) (const char \*meshname, UINT idx, const [VECTOR3](#) \*ofs=0) const  
*Insert or replace a mesh at a specific index location of the vessel's mesh list.*
- UINT [InsertMesh](#) ([MESHHANDLE](#) hMesh, UINT idx, const [VECTOR3](#) \*ofs=0) const  
*Insert or replace a mesh at a specific index location of the vessel's mesh list.*
- bool [DelMesh](#) (UINT idx, bool retain\_anim=false) const  
*Remove a mesh from the vessel's mesh list.*
- bool [ShiftMesh](#) (UINT idx, const [VECTOR3](#) &ofs) const  
*Shift the position of a mesh relative to the vessel's local coordinate system.*
- void [ShiftMeshes](#) (const [VECTOR3](#) &ofs) const  
*Shift the position of all meshes relative to the vessel's local coordinate system.*
- bool [GetMeshOffset](#) (UINT idx, [VECTOR3](#) &ofs) const  
*Returns the mesh offset in the vessel frame.*
- UINT [GetMeshCount](#) () const  
*Number of meshes.*
- [MESHHANDLE](#) [GetMesh](#) ([VISHANDLE](#) vis, UINT idx) const  
*Obtain mesh handle for a vessel mesh.*
- [DEVMESHHANDLE](#) [GetDevMesh](#) ([VISHANDLE](#) vis, UINT idx) const  
*Returns a handle for a device-specific mesh instance.*
- const [MESHHANDLE](#) [GetMeshTemplate](#) (UINT idx) const  
*Obtain a handle for a vessel mesh template.*
- const char \* [GetMeshName](#) (UINT idx) const  
*Obtain mesh file name for an on-demand mesh.*
- [MESHHANDLE](#) [CopyMeshFromTemplate](#) (UINT idx) const  
*Make a copy of one of the vessel's mesh templates.*
- WORD [GetMeshVisibilityMode](#) (UINT idx) const  
*Returns the visibility flags for a vessel mesh.*

- void [SetMeshVisibilityMode](#) (UINT idx, WORD mode) const  
*Set the visibility flags for a vessel mesh.*
- bool [MeshgroupTransform](#) (VISHANDLE vis, const [MESHGROUP\\_TRANSFORM](#) &mt) const  
*Affine transformation of a mesh group.*
- int [MeshModified](#) ([MESHHANDLE](#) hMesh, UINT grp, DWORD modflag)  
*Notifies Orbiter of a change in a mesh group.*

### Animations

- void [RegisterAnimation](#) () const  
*Logs a request for calls to [VESSEL2::clbkAnimate](#).*
- void [UnregisterAnimation](#) () const  
*Unlogs an animation request.*
- UINT [CreateAnimation](#) (double initial\_state) const  
*Create a mesh animation object.*
- bool [DelAnimation](#) (UINT anim) const  
*Delete an existing mesh animation object.*
- [ANIMATIONCOMPONENT\\_HANDLE](#) [AddAnimationComponent](#) (UINT anim, double state0, double state1, MGROUP\_TRANSFORM \*trans, [ANIMATIONCOMPONENT\\_HANDLE](#) parent=NULL) const  
*Add a component (rotation, translation or scaling) to an animation.*
- bool [DelAnimationComponent](#) (UINT anim, [ANIMATIONCOMPONENT\\_HANDLE](#) hAC)  
*Remove a component from an animation.*
- bool [SetAnimation](#) (UINT anim, double state) const  
*Set the state of an animation.*
- UINT [GetAnimPtr](#) ([ANIMATION](#) \*\*anim) const  
*Returns a pointer to the array of animations defined by the vessel.*

### Recording/playback functions

- bool [Recording](#) () const  
*Flag for active recording session.*
- bool [Playback](#) () const  
*Flag for active playback session.*
- void [RecordEvent](#) (const char \*event\_type, const char \*event) const  
*Writes a custom tag to the vessel's articulation data stream during a running recording session.*

## Coordinate transformations

- void [ShiftCentreOfMass](#) (const **VECTOR3** &shift)  
*Register a shift in the centre of mass after a structural change (e.g. stage separation).*
- void [ShiftCG](#) (const **VECTOR3** &shift)  
*Shift the centre of gravity of a vessel.*
- bool [GetSuperstructureCG](#) (**VECTOR3** &cg) const  
*Returns the centre of gravity of the superstructure to which the vessel belongs, if applicable.*
- void [GetRotationMatrix](#) (**MATRIX3** &R) const  
*Returns the current rotation matrix for transformations from the vessel's local frame of reference to the global frame.*
- void [SetRotationMatrix](#) (const **MATRIX3** &R) const  
*Applies a rotation by replacing the vessel's local to global rotation matrix.*
- void [GlobalRot](#) (const **VECTOR3** &rloc, **VECTOR3** &rglob) const  
*Performs a rotation of a direction from the local vessel frame to the global frame.*
- void [HorizonRot](#) (const **VECTOR3** &rloc, **VECTOR3** &rhorizon) const  
*Performs a rotation from the local vessel frame to the current local horizon frame.*
- void [HorizonInvRot](#) (const **VECTOR3** &rhorizon, **VECTOR3** &rloc) const  
*Performs a rotation of a direction from the current local horizon frame to the local vessel frame.*
- void [Local2Global](#) (const **VECTOR3** &local, **VECTOR3** &global) const  
*Performs a transformation from local vessel coordinates to global coordinates.*
- void [Global2Local](#) (const **VECTOR3** &global, **VECTOR3** &local) const  
*Performs a transformation from global to local vessel coordinates.*
- void [Local2Rel](#) (const **VECTOR3** &local, **VECTOR3** &rel) const  
*Performs a transformation from local vessel coordinates to the ecliptic frame centered at the vessel's reference body.*

## Docking port management

See also: [Docking port management](#)

- **DOCKHANDLE** [CreateDock](#) (const **VECTOR3** &pos, const **VECTOR3** &dir, const **VECTOR3** &rot) const  
*Create a new docking port.*
- bool [DelDock](#) (**DOCKHANDLE** hDock) const  
*Delete a previously defined docking port.*
- void [ClearDockDefinitions](#) () const

*Delete all docking ports defined for the vessel.*

- void [SetDockParams](#) (const **VECTOR3** &pos, const **VECTOR3** &dir, const **VECTOR3** &rot) const  
*Set the parameters for the vessel's primary docking port (port 0), or create a new dock if required.*
- void [SetDockParams](#) (**DOCKHANDLE** hDock, const **VECTOR3** &pos, const **VECTOR3** &dir, const **VECTOR3** &rot) const  
*Reset the parameters for a vessel docking port.*
- void [GetDockParams](#) (**DOCKHANDLE** hDock, **VECTOR3** &pos, **VECTOR3** &dir, **VECTOR3** &rot) const  
*Returns the paramters of a docking port.*
- **UINT** [DockCount](#) () const  
*Returns the number of docking ports defined for the vessel.*
- **DOCKHANDLE** [GetDockHandle](#) (**UINT** n) const  
*Returns a handle to a docking port.*
- **OBJHANDLE** [GetDockStatus](#) (**DOCKHANDLE** hDock) const  
*Returns a handle to a docked vessel.*
- **UINT** [DockingStatus](#) (**UINT** port) const  
*Returns a status flag for a docking port.*
- int [Dock](#) (**OBJHANDLE** target, **UINT** n, **UINT** tgtm, **UINT** mode) const  
*Dock to another vessel.*
- bool [Undock](#) (**UINT** n, const **OBJHANDLE** exclude=0) const  
*Release a docked vessel from a docking port.*
- void [SetDockMode](#) (**int** mode) const  
*Set the docking approach mode for all docking ports.*

## Passive attachment management

See also: [Attachment management](#)

- **ATTACHMENTHANDLE** [CreateAttachment](#) (bool toparent, const **VECTOR3** &pos, const **VECTOR3** &dir, const **VECTOR3** &rot, const char \*id, bool loose=false) const  
*Define a new attachment point for a vessel.*
- bool [DelAttachment](#) (**ATTACHMENTHANDLE** attachment) const  
*Delete an attachment point.*
- void [ClearAttachments](#) () const  
*Delete all attachment points defined for the vessel.*

- void `SetAttachmentParams` (`ATTACHMENTHANDLE` attachment, const `VECTOR3` &pos, const `VECTOR3` &dir, const `VECTOR3` &rot) const  
*Reset attachment position and orientation for an existing attachment point.*
- void `GetAttachmentParams` (`ATTACHMENTHANDLE` attachment, `VECTOR3` &pos, `VECTOR3` &dir, `VECTOR3` &rot) const  
*Retrieve the parameters of an attachment point.*
- const char \* `GetAttachmentId` (`ATTACHMENTHANDLE` attachment) const  
*Retrieve attachment identifier string.*
- `OBJHANDLE GetAttachmentStatus` (`ATTACHMENTHANDLE` attachment) const  
*Return the current status of an attachment point.*
- `DWORD AttachmentCount` (bool toparent) const  
*Return the number of child or parent attachment points defined for the vessel.*
- `DWORD GetAttachmentIndex` (`ATTACHMENTHANDLE` attachment) const  
*Return the list index of the vessel's attachment point defined by its handle.*
- `ATTACHMENTHANDLE GetAttachmentHandle` (bool toparent, `DWORD` i) const  
*Return the handle of an attachment point identified by its list index.*
- bool `AttachChild` (`OBJHANDLE` child, `ATTACHMENTHANDLE` attachment, `ATTACHMENTHANDLE` child\_attachment) const  
*Attach a child vessel to an attachment point.*
- bool `DetachChild` (`ATTACHMENTHANDLE` attachment, double vel=0.0) const  
*Break an existing attachment to a child.*

### Exhaust and entry render functions

- `UINT AddExhaust` (`THRUSTER_HANDLE` th, double lscale, double wscale, `SURFHANDLE` tex=0) const  
*Add an exhaust render definition for a thruster.*
- `UINT AddExhaust` (`THRUSTER_HANDLE` th, double lscale, double wscale, double lofs, `SURFHANDLE` tex=0) const  
*Add an exhaust render definition for a thruster with additional offset.*
- `UINT AddExhaust` (`THRUSTER_HANDLE` th, double lscale, double wscale, const `VECTOR3` &pos, const `VECTOR3` &dir, `SURFHANDLE` tex=0) const  
*Add an exhaust render definition for a thruster with explicit reference position and direction.*
- `UINT AddExhaust` (`EXHAUSTSPEC` \*spec)  
*Add an exhaust render definition defined by a parameter structure.*
- bool `DelExhaust` (`UINT` idx) const  
*Removes an exhaust render definition.*

- **DWORD GetExhaustCount () const**  
*Returns the number of exhaust render definitions for the vessel.*
- **bool GetExhaustSpec (UINT idx, double \*lscale, double \*wscale, VECTOR3 \*pos, VECTOR3 \*dir, SURFHANDLE \*tex) const**  
*Returns the parameters of an exhaust definition.*
- **bool GetExhaustSpec (UINT idx, EXHAUSTSPEC \*spec)**  
*Returns the parameters of an exhaust definition in a structure.*
- **double GetExhaustLevel (UINT idx) const**  
*Returns the current level of an exhaust source.*
- **void SetReentryTexture (SURFHANDLE tex, double plimit=6e7, double lscale=1.0, double wscale=1.0) const**  
*Select a previously registered texture to be used for rendering reentry flames.*

### Particle systems

- **PSTREAM\_HANDLE AddParticleStream (PARTICLESTREAMSPEC \*pss, const VECTOR3 &pos, const VECTOR3 &dir, double \*lvl) const**  
*Adds a custom particle stream to a vessel.*
- **PSTREAM\_HANDLE AddExhaustStream (THRUSTER\_HANDLE th, PARTICLESTREAMSPEC \*pss=0) const**  
*Adds an exhaust particle stream to a vessel.*
- **PSTREAM\_HANDLE AddExhaustStream (THRUSTER\_HANDLE th, const VECTOR3 &pos, PARTICLESTREAMSPEC \*pss=0) const**  
*Adds an exhaust particle stream to a vessel.*
- **PSTREAM\_HANDLE AddReentryStream (PARTICLESTREAMSPEC \*pss) const**  
*Adds a reentry particle stream to a vessel.*
- **bool DelExhaustStream (PSTREAM\_HANDLE ch) const**  
*Delete an existing particle stream.*

### Nosewheel-steering and wheel brakes

- **void SetNosewheelSteering (bool activate) const**
- **bool GetNosewheelSteering () const**  
*Returns the activation state of the nose-wheel steering system.*
- **void SetMaxWheelbrakeForce (double f) const**  
*Define the maximum force which can be provided by the vessel's wheel brake system.*
- **void SetWheelbrakeLevel (double level, int which=0, bool permanent=true) const**

*Apply the wheel brake.*

- double [GetWheelbrakeLevel](#) (int which) const

*Returns the current wheel brake level.*

## Beacon management

- void [AddBeacon](#) (BEACONLIGHTSPEC \*bs)

*Add a light beacon definition to a vessel.*

- bool [DelBeacon](#) (BEACONLIGHTSPEC \*bs)

*Remove a beacon definition from the vessel.*

- void [ClearBeacons](#) ()

*Remove all beacon definitions from the vessel.*

- const BEACONLIGHTSPEC \* [GetBeacon](#) (DWORD idx) const

*Returns a pointer to one of the vessel's beacon specifications.*

- LightEmitter \* [AddPointLight](#) (const VECTOR3 &pos, double range, double att0, double att1, double att2, COLOUR4 diffuse, COLOUR4 specular, COLOUR4 ambient) const

*\ name Light emitters*

- LightEmitter \* [AddSpotLight](#) (const VECTOR3 &pos, const VECTOR3 &dir, double range, double att0, double att1, double att2, double umbra, double penumbra, COLOUR4 diffuse, COLOUR4 specular, COLOUR4 ambient) const

*Add a directed spot light source to the vessel.*

- DWORD [LightEmitterCount](#) () const

*Returns the number of light sources defined for the vessel.*

- const LightEmitter \* [GetLightEmitter](#) (DWORD i) const

*Returns a pointer to a light source object identified by index.*

- bool [DelLightEmitter](#) (LightEmitter \*le) const

*Deletes the specified light source from the vessel.*

- void [ClearLightEmitters](#) () const

*Remove all light sources defined for the vessel.*

## File I/O

- void [ParseScenarioLineEx](#) (char \*line, void \*status) const

*Pass a line read from a scenario file to Orbiter for default processing.*

### Obsolete methods

- void **SetEngineLevel** (ENGINETYPE eng, double level) const  
*Set the thrust level for an engine group.*
- void **IncEngineLevel** (ENGINETYPE eng, double dlevel) const  
*Increase or decrease the thrust level for an engine group.*
- double **GetMaxThrust** (ENGINETYPE eng) const
- void **SetMaxThrust** (ENGINETYPE eng, double th) const
- double **GetEngineLevel** (ENGINETYPE eng) const
- double \* **GetMainThrustModPtr** () const
- void **SetExhaustScales** (EXHAUSTTYPE exh, WORD id, double lscale, double wscale) const
- bool **DelThrusterGroup** (THGROUP\_HANDLE &thg, THGROUP\_TYPE thgt, bool delth=false) const  
*Delete a thruster group and (optionally) all associated thrusters.*
- UINT **AddExhaustRef** (EXHAUSTTYPE exh, VECTOR3 &pos, double lscale=-1.0, double wscale=-1.0, VECTOR3 \*dir=0) const
- void **DelExhaustRef** (EXHAUSTTYPE exh, WORD id) const
- void **ClearExhaustRefs** (void) const
- UINT **AddAttExhaustRef** (const VECTOR3 &pos, const VECTOR3 &dir, double wscale=1.0, double lscale=1.0) const
- void **AddAttExhaustMode** (UINT idx, ATTITUDEMODE mode, int axis, int dir) const
- void **ClearAttExhaustRefs** (void) const
- double **GetBankMomentScale** () const  
*Returns the scaling factor for the yaw moment.*
- void **SetBankMomentScale** (double scale) const  
*Sets the scaling factor for the yaw moment.*
- bool **SetNavRecv** (DWORD n, DWORD ch) const  
*Sets the channel of a NAV radio receiver.*
- DWORD **GetNavRecv** (DWORD n) const  
*Returns the current channel setting of a NAV radio receiver.*
- void **SetCOG\_elev** (double h) const  
*Set the altitude of the vessel's centre of gravity over ground level when landed.*
- void **ClearMeshes** () const  
*Remove all mesh definitions for the vessel.*
- void **SetMeshVisibleInternal** (UINT idx, bool visible) const  
*Marks a mesh as visible from internal cockpit view.*
- UINT **RegisterAnimSequence** (double defmeshstate) const
- bool **AddAnimComp** (UINT seq, ANIMCOMP \*comp)
- bool **SetAnimState** (UINT seq, double state)
- void **SaveDefaultState** (FILEHANDLE scn) const

*Causes Orbiter to write default vessel parameters to a scenario file.*

- void [ParseScenarioLine](#) (char \*line, [VESSELSTATUS](#) \*status) const  
*Pass a line read from a scenario file to Orbiter for default processing.*
- static [OBJHANDLE](#) [Create](#) (const char \*name, const char \*classname, const [VESSELSTATUS](#) &status)  
*Vessel creation.*

## Protected Attributes

- Vessel \* [vessel](#)  
*Orbiter internal vessel class.*
- short [flightmodel](#)  
*realism level*
- short [version](#)  
*interface version*

## 16.51.2 Constructor & Destructor Documentation

### 16.51.2.1 VESSEL::VESSEL ([OBJHANDLE](#) *hVessel*, int *fmodel* = 1)

Creates a VESSEL interface instance from a vessel handle.

#### Parameters:

*hVessel* vessel handle  
*fmodel* level of realism requested (0=simple, 1=realistic)

#### Note:

This function creates an interface to an *existing* vessel. It does not create a new vessel. New vessels are created with the [oapiCreateVessel](#) and [oapiCreateVesselEx](#) functions.

The VESSEL constructor (or the constructor of a derived specialised vessel class) will normally be invoked in the ovcInit callback function of a vessel module:

```
class MyVessel: public VESSEL
{
    // MyVessel interface definition
};

DLLCLBK VESSEL *ovcInit (OBJHANDLE hvessel, int flightmodel)
{
    return new MyVessel (hvessel, flightmodel);
}

DLLCLBK void ovcExit (VESSEL *vessel)
{
    delete (MyVessel*)vessel;
}
```

The VESSEL interface instance created in ovcInit should be deleted in ovcExit.

**See also:**

[oapiCreateVessel](#), [oapiCreateVesselEx](#), [ovcInit](#)

### 16.51.3 Member Function Documentation

#### 16.51.3.1 int VESSEL::Version () const [inline]

Returns the version number of the vessel interface class.

**Returns:**

version number

**Note:**

The following interface versions are currently in use:

- class [VESSEL](#): version 0
- class [VESSEL2](#): version 1
- class [VESSEL3](#): version 2

**See also:**

[VESSEL2](#), [VESSEL3](#)

#### 16.51.3.2 const OBJHANDLE VESSEL::GetHandle () const

Returns a handle to the vessel object.

**Returns:**

vessel handle, as passed to the [VESSEL](#) constructor.

**Note:**

The handle is useful for various vessel-related API function calls.

#### 16.51.3.3 bool VESSEL::GetEditorModule (char \**fname*) const

Returns the file name of the DLL containing the vessel's scenario editor extensions.

**Parameters:**

→*fname* module file name

**Returns:**

*true* if the vessel defines an editor module, *false* otherwise.

**Note:**

The vessel's editor module, if it exists, contains extensions for the *Scenario editor* module that allows the user to set vessel-specific parameters (see Doc\ScenarioEditor.pdf).

The string returned by this method is identical to the *EditorModule* entry in the vessel's configuration file.

If the *EditorModule* entry is not found in the configuration file, this method returns *false*.

**16.51.3.4 char\* VESSEL::GetName () const**

Returns the vessel's name.

**Returns:**

Pointer to vessel's name

**See also:**

[GetClassName](#)

**16.51.3.5 char\* VESSEL::GetClassName () const**

Returns the vessel's class name.

**Returns:**

Pointer to vessel's class name.

**See also:**

[GetName](#)

**16.51.3.6 int VESSEL::GetFlightModel () const**

Returns the requested realism level for the flight model.

**Returns:**

Flight model realism level. These values are currently supported:

- 0 = simple
- 1 = realistic

**Note:**

The returned value corresponds to that passed to the [VESSEL](#) constructor. This will normally be the same as the argument of the ovcInit callback function.

The module can use this method to implement different flavours of the flight model (e.g. simplified and realistic), by defining separate sets of parameters (possibly higher fuel-specific impulse and higher thrust ratings in the simplified model, less severe damage limits, etc.)

**See also:**

[ovcInit](#), [GetDamageModel](#)

**16.51.3.7 int VESSEL::GetDamageModel () const**

Returns the current user setting for damage and systems failure simulation.

**Returns:**

Damage modelling flags. The following settings are currently supported:

- 0 = no damage or failures

- 1 = simulate vessel damage and system failures

**Note:**

The return value depends on the user parameter selection in the Launchpad dialog. It does not change during a simulation session and will be the same for all vessels.

Future versions may support more differentiated bit flags to indicate different types of damage and failure simulation.

A vessel implementation should query the damage flag to decide whether to simulate failures.

**See also:**

[GetFlightModel](#)

**16.51.3.8 bool VESSEL::GetEnableFocus () const**

Returns true if the vessel can receive the input focus, false otherwise.

**Returns:**

Focus enabled status.

**Note:**

The vessel can be allowed or prohibited to receive the input focus by using the SetEnableFocus method. The initial state is defined by the EnableFocus setting in the vessel's configuration file. If the entry is missing, the default is true.

Focus-enabled vessels can be selected by the user via the jump vessel dialog (F3).

Once a vessel has received the input focus, all user input via keyboard, mouse and joystick is directed to this vessel.

For some object types, such as jettisoned rocket stages, enabling input focus may not be useful.

**See also:**

[SetEnableFocus](#), [clbkFocusChanged](#), [oapiGetFocusObject](#), [oapiSetFocusObject](#)

**16.51.3.9 void VESSEL::SetEnableFocus (bool *enable*) const**

Enable or disable the vessel's ability to receive the input focus.

**Parameters:**

*enable* focus enabled status: true to allow the vessel to receive input focus, false otherwise.

**Note:**

The initial state is defined by the EnableFocus setting in the vessel's configuration file. If the entry is missing, the default is true.

If the input focus of the current focus vessel is disabled, it will continue to receive user input, until the focus is switched to another vessel.

Focus-enabled vessels can be selected by the user via the jump vessel dialog (F3).

Once a vessel has received the input focus, all user input via keyboard, mouse and joystick is directed to this vessel.

For some object types, such as jettisoned rocket stages, enabling input focus may not be useful.

**See also:**

[GetEnableFocus](#), [clbkFocusChanged](#), [oapiGetFocusObject](#), [oapiSetFocusObject](#)

**16.51.3.10 double VESSEL::GetSize () const**

Returns the vessel's mean radius.

**Returns:**

Vessel mean radius [m].

**Note:**

The value returned is that set by a previous call to SetSize or from the Size entry in the vessel's configuration file.

There is no guarantee that the return value is correlated to the vessel's visual representation. In particular, the size parameter does not change (scale) the visual appearance.

**See also:**

[SetSize](#)

**16.51.3.11 void VESSEL::SetSize (double *size*) const**

Set the vessel's mean radius.

**Parameters:**

*size* vessel mean radius [m].

**Note:**

The size should correspond to the vessel's visual representation, for example the mesh used to show the vessel in the simulation window.

The size parameter is used by Orbiter to determine the camera distance at which the vessel is within visual range of the observer camera. It is also used for calculating various physical parameters.

If SetSize is not called during the vessel setup, the value from the Size entry in the vessel's configuration file is used.

**See also:**

[GetSize](#)

**16.51.3.12 void VESSEL::SetVisibilityLimit (double *vislimit*, double *spotlimit* = -1) const**

Defines the vessel's range of visibility.

**Parameters:**

*vislimit* apparent size limit for vessel visibility

*spotlimit* apparent size limit for vessel "spot" representation.

**Note:**

This function can be used to define the distance up to which a vessel is visible, independent of screen resolution.

The vislimit value is the limiting apparent size (as a fraction of the render window vertical) up to which the vessel is regarded visible. Thus, the vessel is visible if the following condition is satisfied:

$S(d \tan a)^{-1} > vislimit$  where S is the vessel size, d is its camera distance, and a is the camera aperture.

If the defined visibility limit exceeds the distance at which the vessel can be rendered as a mesh at the given screen resolution, it will simply be represented by a circular spot whose size is reduced linearly (to reach zero at the limiting distance).

If the vessel is to be visible beyond its geometric size (e.g. due to light beacons etc.) then the spotlimit value can be used to define the limiting distance due to the vessel's geometry, while vislimit defines the total visibility range including all enhancing factors such as beacons.

spotlimit  $\leq$  vislimit is required. If spotlimit  $< 0$  (default), then spotlimit = vislimit is assumed.

If SetVisibilityLimit is not called, then the default value is vislimit = spotlimit = 1e-3.

#### See also:

[SetSize](#), [SetClipRadius](#)

#### 16.51.3.13 double VESSEL::GetClipRadius () const

Returns the radius of the vessel's circumscribing sphere.

#### Returns:

Radius of the circumscribing sphere of the vessel's visual representation [m].

#### Note:

This parameter describes the radius of the sphere around the vessel that is protected from clipping at the observer camera's near clipping plane. (The near clipping plane defines an area around the view camera within which no objects are rendered. The distance of the near clipping plane cannot be made arbitrarily small for technical reasons.)

By default, the clip radius is identical to the vessel's "Size" parameter. However, the size parameter is correlated to physical vessel properties and may therefore be smaller than the sphere that contains the vessel's complete visual representation. In that case, defining a clip radius that is larger than the size parameter can avoid visual artefacts.

The view camera's near clip plane distance is adjusted so that it does not intersect any nearby vessel's clip radius. However, there is a minimum near clip distance of 2.5m. This means that if the camera approaches a vessel to less than clip radius + 2.5, clipping may still occur.

Visual cockpit meshes are rendered in a separate pass and are not affected by the general near clip distance (they have a separate near clip distance of 10cm).

#### See also:

[SetClipRadius](#), [GetSize](#)

#### 16.51.3.14 void VESSEL::SetAlbedoRGB (const VECTOR3 & albedo) const

Set the average colour distribution reflected by the vessel.

#### Parameters:

*albedo* vessel colour vector (red, green blue), range [0..1] for each component.

#### Note:

The colour passed to this function is currently used to define the "spot" colour with which the vessel is rendered at long distances. It should represent an average colour and brightness of the vessel surface when fully lit.

The values for each of the RGB components should be in the range 0-1.

The default vessel albedo is bright white (1,1,1).

The albedo can be overridden by the AlbedoRGB entry in the vessel's config file.

#### 16.51.3.15 void VESSEL::SetClipRadius (double *rad*) const

Set the radius of the vessel's circumscribing sphere.

**Parameters:**

*rad* Radius of the circumscribing sphere of the vessel's visual representation [m].

**Note:**

This parameter describes the radius of the sphere around the vessel that is protected from clipping at the observer camera's near clipping plane. (The near clipping plane defines an area around the view camera within which no objects are rendered. The distance of the near clipping plane cannot be made arbitrarily small for technical reasons.)

By default, the clip radius is identical to the vessel's "Size" parameter. However, the size parameter is correlated to physical vessel properties and may therefore be smaller than the sphere that contains the vessel's complete visual representation. In that case, defining a clip radius that is larger than the size parameter can avoid visual artefacts.

The view camera's near clip plane distance is adjusted so that it does not intersect any nearby vessel's clip radius. However, there is a minimum near clip distance of 2.5m. This means that if the camera approaches a vessel to less than clip radius + 2.5, clipping may still occur.

Visual cockpit meshes are rendered in a separate pass and are not affected by the general near clip distance (they have a separate near clip distance of 10cm).

Setting *rad* = 0 reverts to the default behaviour of using the vessel's "Size" parameter to determine the clip radius.

**See also:**

[GetClipRadius](#), [SetSize](#)

#### 16.51.3.16 double VESSEL::GetEmptyMass () const

Returns the vessel's empty mass (excluding propellants).

**Returns:**

Vessel empty mass [kg].

**Note:**

The empty mass combines all parts of the vessel except propellant resources defined via [CreatePropellantResource](#).

The empty mass may change during the simulation, often discontinuously, for example as a result of stage separation.

**See also:**

[oapiGetEmptyMass](#), [GetMassDistribution](#), [SetEmptyMass](#), [CreatePropellantResource](#)

**16.51.3.17 void VESSEL::SetEmptyMass (double *m*) const**

Set the vessel's empty mass (excluding propellants).

**Parameters:**

*m* vessel empty mass [kg].

**Note:**

The empty mass combines all parts of the vessel except propellant resources defined via CreatePropellantResource.

Use SetEmptyMass to account for structural changes such as stage or booster separation, but not for fuel consumption, which is done directly by Orbiter.

**See also:**

[GetEmptyMass](#), [SetMassDistribution](#), [oapiSetEmptyMass](#), [CreatePropellantResource](#)

**16.51.3.18 double VESSEL::GetCOG\_elev () const**

Elevation of the vessel's centre of gravity (COG) above ground.

**Returns:**

Distance of COG from vessel ground contact plane [m].

**Note:**

The COG elevation is defined as the normal distance of the vessel's centre of gravity from the ground contact plane defined by its three touchdown points.

By definition, the vessel's centre of gravity coincides with the origin of the local vessel frame.

**See also:**

[GetTouchdownPoints](#), [SetTouchdownPoints](#)

**16.51.3.19 void VESSEL::GetTouchdownPoints (VECTOR3 & *pt1*, VECTOR3 & *pt2*, VECTOR3 & *pt3*) const**

Returns the three points defining the vessel's ground contact plane.

**Parameters:**

*pt1* touchdown point of nose wheel (or equivalent)

*pt2* touchdown point of left main wheel (or equivalent)

*pt3* touchdown point of right main wheel (or equivalent)

**Note:**

The function returns 3 reference points defining the vessel's surface contact points when touched down on a planetary surface (e.g. landing gear).

**See also:**

[SetTouchdownPoints](#), [GetCOG\\_elev](#)

**16.51.3.20 void VESSEL::SetTouchdownPoints (const VECTOR3 & *pt1*, const VECTOR3 & *pt2*, const VECTOR3 & *pt3*) const**

Defines the three points defining the vessel's ground contact plane.

**Parameters:**

- pt1* touchdown point of nose wheel (or equivalent)
- pt2* touchdown point of left main wheel (or equivalent)
- pt3* touchdown point of right main wheel (or equivalent)

**Note:**

The points are the positions at which the vessel's undercarriage (or equivalent) touches the surface, specified in local vessel coordinates.

The order of points is significant since it defines the direction of the normal. The points should be specified such that the cross product  $\text{pt3-pt1} \times \text{pt2-pt1}$  defines the horizon "up" direction for the landed vessel (given a left-handed coordinate system).

Modifying the touchdown points during the simulation while the vessel is on the ground can result in jumps due to instantaneous position changes (infinite acceleration). To avoid this, the touchdown points should be modified gradually by small amounts over time (proportional to simulation time steps).

**See also:**

[GetTouchdownPoints](#), [GetCOG\\_elev](#)

**16.51.3.21 void VESSEL::SetSurfaceFrictionCoeff (double *mu\_lng*, double *mu\_lat*) const**

Set friction coefficients for ground contact.

**Parameters:**

- mu\_lng* friction coefficient in longitudinal direction.
- mu\_lat* friction coefficient in lateral direction.

**Note:**

The coefficients of surface friction define the deceleration forces during sliding or rolling over a surface. *mu\_lng* is the coefficient acting in longitudinal (forward) direction, *mu\_lat* the coefficient acting in lateral (sideways) direction. The friction forces are proportional to the coefficient and the weight of the vessel:

$$\mathbf{F}_{\text{friction}} = \mu \mathbf{G}$$

The higher the coefficient, the faster the vessel will come to a halt.

Typical parameters for a spacecraft equipped with landing wheels would be  $\mu_{\text{lng}} = 0.1$  and  $\mu_{\text{lat}} = 0.5$ . If the vessel hasn't got wheels,  $\mu_{\text{lng}} = 0.5$ .

The coefficients should be adjusted for belly landings when the landing gear is retracted.

The longitudinal and lateral directions are defined by the touchdown points:

$$s_{\text{lng}} = \mathbf{p}_0 - (\mathbf{p}_1 + \mathbf{p}_2)/2, \quad s_{\text{lat}} = \mathbf{p}_2 - \mathbf{p}_1$$

**See also:**

[SetTouchdownPoints](#)

**16.51.3.22 void VESSEL::GetCrossSections (VECTOR3 & *cs*) const**

Returns the vessel's cross sections projected in the direction of the vessel's principal axes.

**Parameters:**

*cs* vector receiving the cross sections of the vessel's projection into the yz, xz and xy planes, respectively [ $\text{m}^2$  ]

**See also:**

[SetCrossSections](#)

**16.51.3.23 void VESSEL::SetCrossSections (const VECTOR3 & *cs*) const**

Defines the vessel's cross-sectional areas, projected in the directions of the vessel's principal axes.

**Parameters:**

*cs* vector of cross-sectional areas of the vessel's projection along the x-axis into yz-plane, along the y-axis into the xz-plane, and along the z-axis into the xy plane, respectively [ $\text{m}^2$  ].

**See also:**

[GetCrossSections](#)

**16.51.3.24 void VESSEL::GetPMI (VECTOR3 & *pmi*) const**

Returns the vessel's mass-normalised principal moments of inertia (PMI).

**Parameters:**

*pmi* Diagonal elements of the vessel's inertia tensor [ $\text{m}^2$  ]

**Note:**

The inertia tensor describes the behaviour of a rigid body under angular acceleration. It is the analog of the body's mass in the linear case.

The values returned by this function are the diagonal elements of the inertia tensor, in the local vessel frame of reference.

Orbiter's definition of PMI is mass-normalised, that is, the values are divided by the total vessel mass. The elements of pmi have the following meaning:

$$\begin{aligned}\text{pmi}_1 &= M^{-1} \int \rho(\vec{r})(\vec{r}_y^2 + \vec{r}_z^2) d\vec{r} \\ \text{pmi}_2 &= M^{-1} \int \rho(\vec{r})(\vec{r}_z^2 + \vec{r}_x^2) d\vec{r} \\ \text{pmi}_3 &= M^{-1} \int \rho(\vec{r})(\vec{r}_x^2 + \vec{r}_y^2) d\vec{r}\end{aligned}$$

Orbiter assumes that off-diagonal elements can be neglected, that is, that the diagonal elements are the principal moments of inertia. This is usually a good approximation when the vessel is sufficiently symmetric with respect to its coordinate frame. Otherwise, a diagonalisation by rotating the local frame may be required.

The shipedit utility in the SDK package allows to calculate the inertia tensor from a mesh, assuming a homogeneous mass distribution.

**See also:**

[SetPMI](#)

#### 16.51.3.25 void VESSEL::SetPMI (const VECTOR3 & *pmi*) const

Set the vessel's mass-normalised principal moments of inertia (PMI).

**Parameters:**

*pmi* pmi Diagonal elements of the vessel's inertia tensor [ $\text{m}^2$  ]

**Note:**

The inertia tensor describes the behaviour of a rigid body under angular acceleration.  
For more information and a definition of the PMI values, see [GetPMI](#).

**See also:**

[GetPMI](#)

#### 16.51.3.26 double VESSEL::GetGravityGradientDamping () const

Returns the vessel's damping coefficient for gravity field gradient-induced torque.

**Returns:**

Torque damping coefficient ( $\geq 0$ )

**Note:**

A nonspherical object in an inhomogeneous gravitational field experiences a torque. Orbiter calculates this torque with

$$\vec{M}_G = \frac{3\mu m}{R^3} (\vec{R}_0 \times \vec{L}\vec{R}_0)$$

where  $\mu = GM$ ,  $G$  is the gravity constant,  $M$  is the reference body mass,  $m$  is the vessel mass,  $R$  is the distance of the vessel to the reference body centre,  $R_0$  is the unit vector towards the reference body, and  $L$  is the mass-normalised inertia tensor (assumed diagonal).

This generates an undamped attitude oscillation in the vessel orbiting the reference body.

Damping may occur due to tidal deformation of the vessel, movement of liquids (fuel) etc. Orbiter allows to introduce a damping term of the form

$$\vec{M}_D = -\alpha \omega_G$$

where  $\omega_G$  is the angular velocity, and  $\alpha = dm r$ , with damping coefficient  $d$ , vessel mass  $m$  and vessel radius  $r$ .

If gravity gradient torque has been disabled in the launchpad dialog, this function always returns 0.

**See also:**

[SetGravityGradientDamping](#), [GetEmptyMass](#), [GetPMI](#)

**16.51.3.27 bool VESSEL::SetGravityGradientDamping (double *damp*) const**

Sets the vessel's damping coefficient for gravity field gradient-induced torque.

**Parameters:**

*damp* Torque damping coefficient.

**Returns:**

true if damping coefficient was applied, false if gravity gradient torque is disabled.

**Note:**

For a definition of the torque experienced by the vessel in an inhomogeneous gravity field, and the damping term that can be applied, see [GetGravityGradientDamping](#).

If gravity gradient torque has been disabled in the launchpad dialog, this function returns false and has no other effect.

**See also:**

[GetGravityGradientDamping](#), [SetEmptyMass](#), [SetPMI](#)

**16.51.3.28 void VESSEL::GetStatus (VESSELSTATUS & *status*) const**

Returns the vessel's current status parameters in a [VESSELSTATUS](#) structure.

**Parameters:**

*status* structure receiving the current vessel status.

**Note:**

The [VESSELSTATUS](#) structure provides only limited information. Applications should normally use [GetStatusEx](#) to obtain a [VESSELSTATUSx](#) structure which contains additional parameters.

**See also:**

[VESSELSTATUS](#), [GetStatusEx](#)

**16.51.3.29 void VESSEL::GetStatusEx (void \* *status*) const**

Returns the vessel's current status parameters in a [VESSELSTATUSx](#) structure (version x >= 2).

**Parameters:**

*status* pointer to a [VESSELSTATUSx](#) structure

**Note:**

This method can be used with any [VESSELSTATUSx](#) interface version supported by Orbiter. Currently only [VESSELSTATUS2](#) is supported.

The version field of the [VESSELSTATUSx](#) structure must be set by the caller prior to calling the method, to tell Orbiter which interface version is required.

In addition, the caller must set the VS\_FUELLIST, VS\_THRUSTLIST and VS\_DOCKINFOLIST bits in the flag field, if the corresponding lists are required. Otherwise Orbiter will not produce these lists.

If VS\_FUELLIST is specified and the fuel field is NULL, Orbiter will allocate memory for the list. The caller is responsible for deleting the list after use. If the fuel field is not NULL, Orbiter assumes that a list of sufficient length to store all propellant resources has been allocated by the caller. The same applies to the thruster and dockinfo lists.

**See also:**

[clbkSetStateEx](#), [DefSetStateEx](#), [VESSELSTATUS2](#)

#### 16.51.3.30 void VESSEL::DefSetState (const VESSELSTATUS \* *status*) const

Set default vessel status parameters.

Invokes Orbiter's vessel state initialisation with the standard status parameters provided via a [VESSELSTATUS](#) structure.

**Parameters:**

*status* structure containing vessel status parameters

**Note:**

The [VESSELSTATUS](#) structure contains only a limited set of parameters. Applications should normally use DefSetStateEx in combination with an extended VESSELSTATUSx structure.

**See also:**

[VESSELSTATUS](#), [DefSetStateEx](#), [GetStatus](#)

#### 16.51.3.31 void VESSEL::DefSetStateEx (const void \* *status*) const

Set default vessel status parameters.

Invokes Orbiter's vessel state initialisation with the standard status parameters provided in a VESSELSTATUSx structure.

**Parameters:**

*status* pointer to a VESSELSTATUSx structure ( $x \geq 2$ ).

**Note:**

*status* must point to a VESSELSTATUSx structure. Currently only [VESSELSTATUS2](#) is supported, but future Orbiter versions may introduce new interfaces.

Typically, this function will be called in the body of an overloaded [VESSEL2::clbkSetStateEx](#) to enable default state initialisation.

**See also:**

[VESSELSTATUS2](#), [GetStatusEx](#), [VESSEL2::clbkSetStateEx](#)

#### 16.51.3.32 DWORD VESSEL::GetFlightStatus () const

Returns a bit flag defining the vessel's current flight status.

**Returns:**

vessel status flags (see notes).

**Note:**

The following flags are currently defined:

- bit 0:
  - 0 = vessel is active (in flight),
  - 1 = vessel is inactive (landed)
- bit 1:
  - 0 = simple vessel (not docked to anything),
  - 1 = part of superstructure, (docked to another vessel)

**16.51.3.33 double VESSEL::GetMass () const**

Returns current (total) vessel mass.

**Returns:**

Current vessel mass [kg].

**Note:**

The returned value does not include any docked or attached vessels.

**See also:**

[SetEmptyMass](#), [GetWeightVector](#), [oapiGetMass](#)

**16.51.3.34 void VESSEL::GetGlobalPos (VECTOR3 & *pos*) const**

Returns the vessel's current position in the global reference frame.

**Parameters:**

*pos* Vector receiving position [**m**]

**Note:**

The global reference frame is the solar barycentric ecliptic system at ecliptic and equinox of J2000.0.

**See also:**

[oapiGetGlobalPos](#), [GetGlobalVel](#), [GetRelativePos](#)

**16.51.3.35 void VESSEL::GetGlobalVel (VECTOR3 & *vel*) const**

Returns the vessel's current velocity in the global reference frame.

**Parameters:**

*vel* Vector receiving velocity [**m/s**]

**Note:**

The global reference frame is the solar barycentric ecliptic system at ecliptic and equinox of J2000.0.

**See also:**

[oapiGetGlobalVel](#), [GetGlobalPos](#), [GetRelativeVel](#)

**16.51.3.36 void VESSEL::GetRelativePos (OBJHANDLE *hRef*, VECTOR3 & *pos*) const**

Returns the vessel's current position with respect to another object.

**Parameters:**

*hRef* reference object handle

*pos* vector receiving position [m]

**Note:**

This function returns the vessel's position relative to the position of the object defined by handle *hRef*. Results are returned in the ecliptic frame (ecliptic and equinox of J2000.0).

**See also:**

[oapiGetRelativePos](#), [GetRelativeVel](#), [GetGlobalPos](#)

**16.51.3.37 void VESSEL::GetRelativeVel (OBJHANDLE *hRef*, VECTOR3 & *vel*) const**

Returns the vessel's current velocity relative to another object.

**Parameters:**

*hRef* reference object handle

*vel* vector receiving velocity [m/s]

**Note:**

This function returns the vessel's velocity relative to the velocity of the object defined by handle *hRef*. Results are returned in the ecliptic frame (ecliptic and equinox of J2000.0).

**See also:**

[oapiGetRelativeVel](#), [GetGlobalVel](#), [GetRelativePos](#)

**16.51.3.38 void VESSEL::GetAngularVel (VECTOR3 & *avel*) const**

Returns the vessel's current angular velocity components around its principal axes.

**Parameters:**

→ *avel* vector receiving angular velocity components [rad/s]

**Note:**

The returned vector contains the angular velocities  $\omega_x, \omega_y, \omega_z$  around the vessel's x, y and z axes, in the rotating vessel frame.

Because the change of the angular velocity components is governed by Euler's coupled differential equations of rigid body motion, the values can fluctuate between the axes even if no torque is acting on the vessel.

**See also:**

[SetAngularVel](#), [GetAngularAcc](#)

**16.51.3.39 void VESSEL::GetAngularAcc (VECTOR3 & *aacc*) const**

Returns the vessel's current angular acceleration components around its principal axes.

**Parameters:**

→ *aacc* angular acceleration [rad/s<sup>2</sup>]

**Note:**

The returned vector contains the angular accelerations  $\partial\omega_x/\partial t, \partial\omega_y/\partial t, \partial\omega_z/\partial t$  around the vessel's x, y and z axes, in the rotating vessel frame.

**See also:**

[GetAngularVel](#), [GetAngularMoment](#)

**16.51.3.40 void VESSEL::GetLinearMoment (VECTOR3 & *F*) const**

Returns the linear force vector currently acting on the vessel.

**Parameters:**

→ *F* force vector in vessel coordinates [N]

**Note:**

The returned vector is the vector sum of all forces (gravity, thrust, aerodynamic forces, etc.) currently acting on the vessel.

**See also:**

[GetAngularMoment](#)

**16.51.3.41 void VESSEL::GetAngularMoment (VECTOR3 & *amom*) const**

Returns the sum of angular moments currently acting on the vessel.

**Parameters:**

→ *amom* angular moment [Nm]

**Note:**

Given all force components  $\mathbf{F}_i$  acting on the vessel at positions  $\mathbf{r}_i$ , the angular moment is defined as

$$\vec{M} = \sum_i \vec{F}_i \times \vec{r}_i$$

(note the left-handed reference frame in the order of operands for the cross product).

**See also:**

[GetLinearMoment](#)

**16.51.3.42 void VESSEL::SetAngularVel (const VECTOR3 & *avel*) const**

Applies new angular velocity to the vessel.

**Parameters:**

*avel* vector containing the new angular velocity components [rad/s]

**Note:**

The input vector defines the angular velocities around the vessel's x, y and z axes. They refer to the rotating vessel frame.

**See also:**

[GetAngularVel](#)

**16.51.3.43 void VESSEL::GetGlobalOrientation (VECTOR3 & *arot*) const**

Returns the Euler angles defining the vessel's orientation.

**Parameters:**

*arot* vector receiving the three Euler angles [rad]

**Note:**

The components of the returned vector  $\text{arot} = (\alpha, \beta, \gamma)$  are the angles of rotation [rad] around the x,y,z axes in the global (ecliptic) frame to produce the rotation matrix  $\mathbf{R}$  for mapping from the vessel's local frame of reference to the global frame of reference:

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & \sin \alpha \\ 0 & -\sin \alpha & \cos \alpha \end{bmatrix} \begin{bmatrix} \cos \beta & 0 & -\sin \beta \\ 0 & 1 & 0 \\ \sin \beta & 0 & \cos \beta \end{bmatrix} \begin{bmatrix} \cos \gamma & \sin \gamma & 0 \\ -\sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

**See also:**

[SetGlobalOrientation](#), [GetRotationMatrix](#)

**16.51.3.44 void VESSEL::SetGlobalOrientation (const VECTOR3 & *arot*) const**

Sets the vessel's orientation via Euler angles.

**Parameters:**

*arot* vector containing the set of Euler angles [rad]

**Note:**

Given the rotation matrix  $\mathbf{R}$  which transforms from the local (vessel) frame to the global (ecliptic) reference frame, the Euler angles expected by this method are defined as

$$\begin{aligned}\alpha &= \text{atan2}(R_{23}, R_{33}) \\ \beta &= \text{asin}(R_{13}) \\ \gamma &= \text{atan2}(R_{12}, R_{11})\end{aligned}$$

**See also:**

[GetGlobalOrientation](#), [SetRotationMatrix](#)

**16.51.3.45 bool VESSEL::GroundContact () const**

Returns a flag indicating contact with a planetary surface.

**Returns:**

true indicates ground contact (at least one of the vessel's touchdown reference points is in contact with a planet surface).

**See also:**

[SetTouchdownPoints](#)

**16.51.3.46 bool VESSEL::OrbitStabilised () const**

Flag indicating whether orbit stabilisation is used for the vessel at the current time step.

**Returns:**

true indicates that the vessel's state is currently updated by using the stabilisation algorithm, which calculates the osculating elements with respect to the primary gravitational source, and treats all additional forces as perturbations.

**Note:**

A vessel switches to orbit stabilisation only if the user has enabled it in the launchpad dialog, and the user-defined perturbation and time step limits are currently satisfied.

Stabilised mode reduces the effect of deteriorating orbits due to accumulating numerical errors in the state vector propagation, but is limited in handling multiple gravitational sources.

**See also:**

[GetElements](#)

**16.51.3.47 bool VESSEL::NonsphericalGravityEnabled () const**

Flag for nonspherical gravity perturbations.

Indicates whether the vessel considers gravity field perturbations due to nonspherical planet shapes when updating its state vectors for the current time step.

**Returns:**

true indicates that gravity perturbations due to nonspherical planet shapes are taken into account.

**Note:**

This function will always return false if the user has disabled the "Nonspherical gravity sources" option in the Launchpad dialog.

If the user has enabled orbit stabilisation in the Launchpad, this function may sometimes return false during high time compression, even if the nonspherical option has been selected. In such situations Orbiter can exclude nonspherical perturbations to avoid numerical instabilities.

**See also:**

[GetWeightVector](#)

**16.51.3.48 DWORD VESSEL::GetADCtrlMode () const**

Returns aerodynamic control surfaces currently under manual control.

**Returns:**

Bit flags defining the current address mode for aerodynamic control surfaces.

**Note:**

The input mode defines which types of control surfaces can be manually controlled by the user.

The returned control mode contains bit flags as follows:

- bit 0: elevator enabled/disabled
- bit 1 rudder enabled/disabled
- bit 2 ailerons enabled/disabled

Therefore, mode=0 indicates control surfaces disabled, mode=7 indicates fully enabled.

Some vessel types may support not all, or not any, types of control surfaces.

**See also:**

[SetADCtrlMode](#), [CreateControlSurface](#), [CreateControlSurface2](#), [GetControlSurfaceLevel](#), [SetControlSurfaceLevel](#)

**16.51.3.49 void VESSEL::SetADCtrlMode (DWORD *mode*) const**

Configure manual input mode for aerodynamic control surfaces.

**Parameters:**

*mode* bit flags defining the address mode for aerodynamic control surfaces (see notes)

**Note:**

The mode parameter contains bit flags as follows:

- bit 0: enable/disable elevator
- bit 1: enable/disable rudder
- bit 2 enable/disable ailerons

Therefore, use mode = 0 to disable all control surfaces, mode = 7 to enable all control surfaces.

**See also:**

[GetADCrlMode](#), [CreateControlSurface](#), [CreateControlSurface2](#), [GetControlSurfaceLevel](#), [SetControlSurfaceLevel](#)

**16.51.3.50 bool VESSEL::ActivateNavmode (int mode)**

Activates one of the automated orbital navigation modes.

**Parameters:**

*mode* navigation mode identifier (see [Navigation mode identifiers](#))

**Returns:**

*true* if the specified navigation mode could be activated, *false* if not available or active already.

**Note:**

Navmodes are high-level navigation modes which involve e.g. the simultaneous and timed engagement of multiple attitude thrusters to get the vessel into a defined state. Some navmodes terminate automatically once the target state is reached (e.g. killrot), others remain active until explicitly terminated (hlevel). Navmodes may also terminate if a second conflicting navmode is activated.

**See also:**

[Navigation mode identifiers](#), [DeactivateNavmode](#), [ToggleNavmode](#), [GetNavmodeState](#)

**16.51.3.51 bool VESSEL::DeactivateNavmode (int mode)**

Deactivates an automated orbital navigation mode.

**Parameters:**

*mode* navigation mode identifier (see [Navigation mode identifiers](#))

**Returns:**

*true* if the specified navigation mode could be deactivated, *false* if not available or inactive already.

**See also:**

[Navigation mode identifiers](#), [ActivateNavmode](#), [ToggleNavmode](#), [GetNavmodeState](#)

**16.51.3.52 bool VESSEL::ToggleNavmode (int mode)**

Toggles a navigation mode on/off.

**Parameters:**

*mode* navigation mode identifier (see [Navigation mode identifiers](#))

**Returns:**

*true* if the specified navigation mode could be changed, *false* if it remains unchanged.

**See also:**

[Navigation mode identifiers](#), [ActivateNavmode](#), [DeactivateNavmode](#), [GetNavmodeState](#)

**16.51.3.53 bool VESSEL::GetNavmodeState (int mode)**

Returns the current active/inactive state of a navigation mode.

**Parameters:**

*mode* navigation mode identifier (see [Navigation mode identifiers](#))

**Returns:**

*true* if the specified navigation mode is active, *false* otherwise.

**See also:**

[Navigation mode identifiers](#), [ActivateNavmode](#), [DeactivateNavmode](#), [ToggleNavmode](#)

**16.51.3.54 const OBJHANDLE VESSEL::GetGravityRef () const**

Returns a handle to the main contributor of the gravity field at the vessel's current position.

**Returns:**

Handle for gravity reference object.

**Note:**

All parameters calculated by functions in this section refer to the gravity reference object, unless explicitly stated otherwise.

**16.51.3.55 OBJHANDLE VESSEL::GetElements (ELEMENTS & el, double & mjd\_ref) const**

Returns osculating orbital elements.

Calculates the set of osculating elements at the current time with respect to the dominant gravitational source.

**Parameters:**

→ *el* current osculating elements relative to dominant gravity source, in ecliptic frame of reference.

→ *mjd\_ref* reference date (in Modified Julian Date format) to which the returned el.L (mean longitude) value refers.

#### Returns:

Handle of reference object. NULL indicates failure (no elements available).

#### Note:

This method will return the mean longitude at a fixed reference date, so the value will not change over time, unless the orbit itself changes.

For extended functionality, see version 2 of GetElements.

#### See also:

[Basics of orbital mechanics](#), [ELEMENTS](#), [GetElements\(OBJHANDLE,ELEMENTS&,ORBITPARAM\\*,double,int\)const](#), [SetElements](#)

### 16.51.3.56 bool VESSEL::GetElements (OBJHANDLE *hRef*, ELEMENTS & *el*, ORBITPARAM \* *prm* = 0, double *mjd\_ref* = 0, int *frame* = FRAME\_ECL) const

Returns osculating elements and additional orbit parameters.

Returns the current osculating elements for the vessel. This version has an extended functionality: it allows to specify an arbitrary celestial body as reference object, an arbitrary reference time, and can return elements either in the ecliptic or equatorial frame of reference.

#### Parameters:

*hRef* reference body handle

*el* current osculating elements relative to hRef

*prm* additional orbital parameters

*mjd\_ref* reference data (in Modified Julian Date format) to which the el.L (mean longitude) value refers.

*frame* orientation of reference frame (see notes)

#### Returns:

Currently always true.

#### Note:

For an overview of orbital parameters, see [Basics of orbital mechanics](#).

This version returns the elements with respect to an arbitrary celestial body, even if that body is not the main source of the gravity field acting on the vessel. If hRef==NULL, the default reference body is used.

If the prm pointer is not set to NULL, the **ORBITPARAM** structure it points to will be filled with additional orbital parameters derived from the primary elements.

All parameters returned in the prm structure refer to the current date, rather than the reference date mjd\_ref. Therefore, the values of el.L and prm->MnL can be different.

Unlike GetElements(ELEMENTS&,double&), mjd\_ref is a user-provided input parameter which specifies to which date the returned el.L (mean longitude) value will refer. An exception is mjd\_ref = 0, which is interpreted as the current time (equivalent to mjd\_ref = [oapiGetSimMJD\(\)](#) ).

The frame parameter can be set to one of the following:

- FRAME\_ECL: returned elements are expressed in the ecliptic frame (epoch J2000).

- FRAME\_EQU: returned elements are expressed in the equatorial frame of the reference object (hRef).

**See also:**

[Basics of orbital mechanics](#), [ELEMENTS](#), [ORBITPARAM](#), [GetElements\(ELEMENTS&,double&\)const](#), [SetElements](#)

### 16.51.3.57 bool VESSEL::SetElements (OBJHANDLE *hRef*, const ELEMENTS & *el*, ORBITPARAM \* *prm* = 0, double *mjd\_ref* = 0, int *frame* = FRAME\_ECL) const

Set vessel state (position and velocity) by means of a set of osculating orbital elements.

**Parameters:**

*hRef* reference body handle  
*el* set of elements to be applied  
*prm* secondary orbital parameters  
*mjd\_ref* reference date (in Modified Julian Date format) to which the el.L (mean longitude) value refers  
*frame* orientation of reference frame (see notes)

**Returns:**

If the vessel position resulting from applying the elements would be located below the surface of the reference body, the method does nothing and returns false. Otherwise it returns true.

**Note:**

This method resets the vessel's position and velocity according to the specified orbital elements. If the prm pointer is not set to NULL, the [ORBITPARAM](#) structure it points to will be filled with secondary orbital parameters derived from the primary elements el. Note that this is an output parameter, i.e. the resulting vessel state will not be influenced by initialising this structure prior to the function call.

All parameters returned in the prm structure refer to the current date, rather than the reference date mjd\_ref. Therefore, the values of el.L and prm->MnL can be different.

The elements can be supplied either in terms of the ecliptic frame (frame = FRAME\_ECL) or in the equatorial frame of the reference body (frame = FRAME\_EQU).

mjd\_ref is an input parameter which defines the date to which the el.L (mean longitude) value refers. An exception is mjd\_ref = 0, which is interpreted as the current time (equivalent to mjd\_ref = [oapiGetSimMJD\(\)](#) ).

Calling SetElements will always put a vessel in freeflight mode, even if it had been landed before. Currently, SetElements doesn't check for validity of the provided elements. Setting invalid elements, or elements which put the vessel below a planetary surface will produce undefined results.

**See also:**

[Basics of orbital mechanics](#), [ELEMENTS](#), [ORBITPARAM](#), [GetElements\(ELEMENTS&,double&\)const](#), [GetElements\(OBJHANDLE,ELEMENTS&,ORBITPARAM\\*,double,int\)const](#)

**16.51.3.58 OBJHANDLE VESSEL::GetSMi (double & *smi*) const**

Returns the magnitude of the semi-minor axis of the current osculating orbit.

**Parameters:**

→ *smi* semi-minor axis [m]

**Returns:**

Handle of reference object, relative to which the orbit is calculated. NULL indicates failure (no orbit information available)

**Note:**

The semi-minor axis is the smallest semi-diameter of the orbit ellipse (see [Basics of orbital mechanics](#)).

**See also:**

[Basics of orbital mechanics](#), [ELEMENTS](#), [ORBITPARAM](#), [GetElements](#)

**16.51.3.59 OBJHANDLE VESSEL::GetArgPer (double & *arg*) const**

Returns argument of periapsis of the current osculating orbit.

**Parameters:**

→ *arg* argument of periapsis for current orbit [rad]

**Returns:**

Handle of reference body, relative to which the orbit is calculated. NULL indicates failure (no orbit information available)

**Note:**

The argument of periapsis is the angle between periapsis and the ascending node (see [The orbit in space](#)).

**See also:**

[Basics of orbital mechanics](#), [ELEMENTS](#), [ORBITPARAM](#), [GetPeDist](#), [GetApDist](#), [GetElements](#)

**16.51.3.60 OBJHANDLE VESSEL::GetPeDist (double & *pedist*) const**

Returns the periapsis distance of the current osculating orbit.

**Parameters:**

→ *pedist* periapsis distance [m]

**Returns:**

Handle of reference body, relative to which the orbit is calculated. NULL indicates failure (no orbit information available)

**Note:**

The periapsis distance is the smallest radius of the orbit (see [Basics of orbital mechanics](#)).

**See also:**

[Basics of orbital mechanics](#), [ELEMENTS](#), [ORBITPARAM](#), [GetApDist](#), [GetArgPer](#), [GetElements](#)

**16.51.3.61 const OBJHANDLE VESSEL::GetApDist (double & *apdist*) const**

Returns the apoapsis distance of the current osculating orbit.

**Parameters:**

→ *apdist* apoapsis distance [m]

**Returns:**

Handle of reference body, relative to which the orbit is calculated. NULL indicates failure (no orbit information available)

**Note:**

the apoapsis distance is the largest radius of the orbit (see [Basics of orbital mechanics](#)).

**See also:**

[Basics of orbital mechanics](#), [ELEMENTS](#), [ORBITPARAM](#), [GetPeDist](#), [GetArgPer](#), [GetElements](#)

**16.51.3.62 const OBJHANDLE VESSEL::GetSurfaceRef () const**

Returns a handle to the surface reference object (planet or moon).

**Returns:**

Surface reference object handle

**Note:**

The surface reference is the planet or moon whose surface is closest to the current vessel position. All methods in this group refer to this celestial body.

**16.51.3.63 double VESSEL::GetAltitude () const**

Returns the altitude above the surface of the surface reference body.

**Returns:**

Altitude [m]

**Note:**

Currently all celestial bodies are assumed to be spheres. This method therefore returns the distance to the centre of the reference body minus the reference body radius.

The reference body is the planet or moon whose surface is closest to the current vessel position (i.e. the body with minimal altitude).

**See also:**

[GetSurfaceRef](#)

**16.51.3.64 double VESSEL::GetPitch () const**

Returns the current pitch angle with respect to the local horizon.

**Returns:**

pitch angle [rad]

**Note:**

The pitch angle  $p$  is defined as

$$p = \frac{\pi}{2} - q$$

where  $q$  is the angle between the vessel's positive z axis (forward direction) and the normal of the local horizon.

**See also:**

[GetSurfaceRef](#), [GetBank](#), [GetYaw](#)

**16.51.3.65 double VESSEL::GetBank () const**

Returns the current bank (roll) angle with respect to the local horizon.

**Returns:**

bank angle [rad]

**Note:**

The bank angle  $b$  is defined as the angle between the vessel's positive y axis (up direction) and the projection of the normal of the local horizon into the x-y plane.

**See also:**

[GetSurfaceRef](#), [GetPitch](#), [GetYaw](#)

**16.51.3.66 double VESSEL::GetYaw () const**

Returns the current yaw angle with respect to the local horizon.

**Returns:**

yaw angle [rad]

**Note:**

The yaw angle  $y$  is defined as the angle between the the projection of the vessel's positive z axis (forward direction) into the horizon plane, and the local horizon "north" direction.

**See also:**

[GetSurfaceRef](#), [GetPitch](#), [GetBank](#)

**16.51.3.67 OBJHANDLE VESSEL::GetEquPos (double & *longitude*, double & *latitude*, double & *radius*) const**

Returns vessel's current equatorial position with respect to the closest planet or moon.

**Parameters:**

- *longitude* longitude coordinate [rad]
- *latitude* latitude coordinate [rad]
- *radius* distance from planet centre [m]

**Returns:**

Handle to reference body to which the parameters refer. NULL indicates failure (no reference body available).

**See also:**

[GetSurfaceRef](#)

**16.51.3.68 const OBJHANDLE VESSEL::GetAtmRef () const**

Returns a handle to the reference body for atmospheric calculations.

**Returns:**

Handle for the celestial body whose atmosphere the vessel is currently moving through, or NULL if the vessel is not inside an atmosphere.

**See also:**

[GetAtmTemperature](#), [GetAtmDensity](#), [GetAtmPressure](#)

**16.51.3.69 double VESSEL::GetAtmTemperature () const**

Returns ambient atmospheric temperature at current vessel position.

**Returns:**

Ambient temperature [K] at current vessel position.

**Note:**

This function returns 0 if the vessel is outside all planetary atmospheric hulls, as defined by the planets' AtmAltLimit parameters.

**See also:**

[GetAtmDensity](#), [GetAtmPressure](#), [GetAtmRef](#)

**16.51.3.70 double VESSEL::GetAtmDensity () const**

Returns atmospheric density at current vessel position.

**Returns:**

Atmospheric density [kg/m<sup>3</sup>] at current vessel position.

**Note:**

This function returns 0 if the vessel is outside all planetary atmospheric hulls, as defined by the planets' AtmAltLimit parameters.

**See also:**

[GetAtmPressure](#), [GetAtmTemperature](#), [GetAtmRef](#)

**16.51.3.71 double VESSEL::GetAtmPressure () const**

Returns static atmospheric pressure at current vessel position.

**Returns:**

Static atmospheric pressure [Pa] at current vessel position.

**Note:**

This function returns 0 if the vessel is outside all planetary atmospheric hulls, as defined by the planets' AtmAltLimit parameters.

**See also:**

[GetDynPressure](#), [GetAtmDensity](#), [GetAtmTemperature](#), [GetAtmRef](#)

**16.51.3.72 double VESSEL::GetDynPressure () const**

Returns the current dynamic pressure for the vessel.

**Returns:**

Current vessel dynamic pressure [Pa].

**Note:**

The dynamic pressure is defined as

$$q = \frac{1}{2} \rho V^2$$

with density  $\rho$  and airflow velocity  $V$ . Dynamic pressure is an important aerodynamic parameter.

**See also:**

[GetAtmPressure](#), [GetAtmRef](#)

**16.51.3.73 double VESSEL::GetMachNumber () const**

Returns the vessel's current Mach number.

**Returns:**

Mach number - the ratio of current freestream airflow velocity over speed of sound.

**Note:**

The speed of sound depends on several parameters, e.g. atmospheric composition and temperature. The Mach number can therefore vary even if the airspeed is constant.

**See also:**

[GetAirspeed](#), [GetAtmRef](#)

**16.51.3.74 double VESSEL::GetGroundspeed () const**

Returns magnitude of the ground speed vector.

**Returns:**

Magnitude of ground speed velocity vector [m/s]

**Note:**

The ground speed vector is defined as the ship's velocity vector in the rotating frame of the celestial reference body.

This function returns the length of the vector returned by GetGroundspeedVector.

**See also:**

[GetGroundspeedVector](#), [GetAirspeed](#), [GetAirspeedVector](#), [GetMachNumber](#), [GetAtmRef](#)

**16.51.3.75 bool VESSEL::GetGroundspeedVector (REFFRAME *frame*, VECTOR3 & *v*) const**

Returns the vessel's ground speed vector.

**Parameters:**

← *frame* frame of reference for returned vector.

→ *v* ground speed vector on exit [m/s]

**Returns:**

Status flag (*false* indicates error). Error conditions: invalid *frame* parameter, or ground speed data could not be obtained.

**Note:**

This method returns the ground speed vector in the requested frame of reference. The ground speed vector is defined as the vessel's velocity vector measured in the rotating frame of the celestial reference body.

Valid entries for *frame* are

- FRAME\_GLOBAL: Returns velocity vector in the global frame of reference

- FRAME\_LOCAL: Returns velocity vector in the vessel's local frame of reference
- FRAME\_REFLOCAL: Returns velocity vector in the celestial reference body's local frame of reference
- FRAME\_HORIZON: Returns velocity vector in the local horizon frame (x = longitudinal component, y = vertical component, z = latitudinal component)

**See also:**

[GetGroundspeed](#), [GetAirspeed](#), [GetAirspeedVector](#), [GetAtmRef](#)

**16.51.3.76 double VESSEL::GetAirspeed () const**

Returns magnitude of the true airspeed vector.

**Returns:**

Magnitude of airspeed velocity vector [m/s]

**Note:**

The airspeed vector is defined as the ship's velocity vector relative to the velocity vector of the surrounding freestream airflow.

This function returns the length of the vector returned by [GetAirspeedVector\(\)](#).

**See also:**

[GetAirspeedVector](#), [GetGroundspeed](#), [GetGroundspeedVector](#), [GetMachNumber](#), [GetAtmRef](#)

**16.51.3.77 bool VESSEL::GetAirspeedVector (REFFRAME *frame*, VECTOR3 & *v*) const**

Returns the vessel's true airspeed vector.

**Parameters:**

$\leftarrow \text{frame}$  frame of reference for returned vector.  
 $\rightarrow v$  airspeed vector on exit [**m/s**]

**Returns:**

Status flag (*false* indicates error). Error conditions: invalid *frame* parameter, or ground speed data could not be obtained.

**Note:**

This method returns the true airspeed vector in the requested frame of reference. The ground airvector is defined as the vessel's velocity vector with respect to the surrounding freestream air flow.

If the vessel is not within an a planetary atmosphere, the returned vector is equal to the groundspeed vector.

Valid entries for *frame* are

- FRAME\_GLOBAL: Returns velocity vector in the global frame of reference
- FRAME\_LOCAL: Returns velocity vector in the vessel's local frame of reference
- FRAME\_REFLOCAL: Returns velocity vector in the celestial reference body's local frame of reference

- FRAME\_HORIZON: Returns velocity vector in the local horizon frame ( $x =$  longitudinal component,  $y =$  vertical component,  $z =$  latitudinal component)

See also:

[GetAirspeed](#), [GetGroundspeed](#), [GetGroundspeedVector](#), [GetAtmRef](#)

#### 16.51.3.78 bool VESSEL::GetHorizonAirspeedVector (VECTOR3 & $v$ ) const

Returns the airspeed vector in local horizon coordinates.

##### Deprecated

This method has been replaced by [VESSEL::GetAirspeedVector](#)

#### 16.51.3.79 bool VESSEL::GetShipAirspeedVector (VECTOR3 & $v$ ) const

Returns the airspeed vector in vessel coordinates.

##### Deprecated

This method has been replaced by [VESSEL::GetAirspeedVector](#)

#### 16.51.3.80 double VESSEL::GetAOA () const

Returns the current angle of attack.

##### Returns:

AOA (angle of attack) value [rad] in the range  $-Pi \dots +Pi$ .

##### Note:

The AOA value is defined as the angle between the vessel's positive z axis and the flight path direction, projected into the yz-plane of the vessel's local coordinate system.

See also:

[GetSlipAngle](#)

#### 16.51.3.81 double VESSEL::GetSlipAngle () const

Returns the lateral (yaw) angle between the velocity vector and the vessel's longitudinal axis.

##### Returns:

slip angle [rad] in the range  $-Pi \dots +Pi$ .

##### Note:

The slip angle is defined as the angle between the vessel's positive z axis and the flight path direction, projected into the xz-plane of the vessel's local coordinate system.

See also:

[GetAOA](#)

### 16.51.3.82 void VESSEL::CreateAirfoil (AIRFOIL\_ORIENTATION *align*, const VECTOR3 & *ref*, AirfoilCoeffFunc *cf*, double *c*, double *S*, double *A*) const

Creates a new airfoil and defines its aerodynamic properties.

**Parameters:**

- align* orientation of the lift vector (LIFT\_VERTICAL or LIFT\_HORIZONTAL)
- ref* centre of pressure in vessel coordinates [m]
- cf* pointer to coefficient callback function (see notes)
- c* airfoil chord length [m]
- S* wing area [ $m^2$  ]
- A* wing aspect ratio

**Note:**

A vessel can define multiple airfoils (for wings, main body, tail stabilisers, etc.). In general, it should define at least one vertical and one horizontal component.

Airfoil definitions for wings and horizontal stabilisers set *align* to LIFT\_VERTICAL. Vertical stabilisers (vertical tail fin, etc.) set *align* to LIFT\_HORIZONTAL.

The centre of pressure is the point at which the lift and drag forces generated by the airfoil are applied to the vessel. Together with the moment coefficient it defines the aerodynamic stability of the vessel. Usually the CoP will be aft of the CG, and the moment coefficient will have a negative slope around the trim angle of attack.

The *AirfoilCoeffFunc* is a callback function which must be supplied by the module. It calculates the lift, moment and drag coefficients for the airfoil. It has the following interface:

```
void AirfoilCoeffFunc (double aoa, double M, double Re,
                      double *cl, double *cm, double *cd)
```

and returns the lift coefficient (*cl*), moment coefficient (*cm*) and drag coefficient (*cd*) as a function of angle of attack *aoa* [rad], Mach number *M* and Reynolds number *Re*. Note that *aoa* can range over the full circle (-pi to pi). For vertical lift components, *aoa* is the pitch angle of attack (a), while for horizontal components it is the yaw angle of attack (b).

If the wing area *S* is set to 0, then Orbiter uses the projected vessel cross sections to define a reference area. Let (*v<sub>x</sub>* , *v<sub>y</sub>* , *v<sub>z</sub>* ) be the unit vector of freestream air flow in vessel coordinates. Then the reference area is calculated as *S* = *v<sub>z</sub>* *C<sub>z</sub>* + *v<sub>y</sub>* *C<sub>y</sub>* for a LIFT\_VERTICAL airfoil, and as *S* = *v<sub>z</sub>* *C<sub>z</sub>* + *v<sub>x</sub>* *C<sub>x</sub>* for a LIFT\_HORIZONTAL airfoil, where *C<sub>x</sub>* , *C<sub>y</sub>* , *C<sub>z</sub>* are the vessel cross-sections in x, y and z direction, respectively.

The wing aspect ratio is defined as *A* = *b*<sup>2</sup> /*S* with wing span *b*.

A vessel should typically define its airfoils in the [VESSEL2::clbkSetClassCaps](#) callback function. If no airfoils are defined, Orbiter will fall back to its legacy drag calculation, using the cw coefficients defined in [SetCW](#). Legacy lift calculation is no longer supported.

For more details, see the Programmer's Guide.

**See also:**

[CreateAirfoil2](#), [CreateAirfoil3](#), [EditAirfoil](#), [DelAirfoil](#)

### 16.51.3.83 AIRFOILHANDLE VESSEL::CreateAirfoil2 (AIRFOIL\_ORIENTATION *align*, const VECTOR3 & *ref*, AirfoilCoeffFunc *cf*, double *c*, double *S*, double *A*) const

Creates a new airfoil and defines its aerodynamic properties.

**Parameters:**

*align* orientation of the lift vector (LIFT\_VERTICAL or LIFT\_HORIZONTAL)  
*ref* centre of pressure in vessel coordinates [m]  
*cf* pointer to coefficient callback function (see notes)  
*c* airfoil chord length [m]  
*S* wing area [ $\text{m}^2$ ]  
*A* wing aspect ratio

**Returns:**

Handle for the new airfoil.

**Note:**

This method is identical to [CreateAirfoil](#), but returns a handle which can be used to identify the airfoil later.

**See also:**

[CreateAirfoil](#), [CreateAirfoil3](#), [EditAirfoil](#), [DelAirfoil](#)

**16.51.3.84 AIRFOILHANDLE VESSEL::CreateAirfoil3 (AIRFOIL\_ORIENTATION *align*, const VECTOR3 & *ref*, AirfoilCoeffFuncEx *cf*, void \* *context*, double *c*, double *S*, double *A*) const**

Creates a new airfoil and defines its aerodynamic properties.

**Parameters:**

*align* orientation of the lift vector (LIFT\_VERTICAL or LIFT\_HORIZONTAL)  
*ref* centre of pressure in vessel coordinates [m]  
*cf* pointer to coefficient callback function (see notes)  
*context* pointer to data block passed to cf callback function  
*c* airfoil chord length [m]  
*S* wing area [ $\text{m}^2$ ]  
*A* wing aspect ratio

**Returns:**

Handle for the new airfoil.

**Note:**

This method is an extension to [CreateAirfoil2](#), using a more versatile coefficient callback function. AirfoilCoeffFuncEx has the following interface:

```
void AirfoilCoeffFuncEx (VESSEL *v, double aoa, double M, double Re,
                        void *context, double *cl, double *cm, double *cd)
```

where *v* is a pointer to the calling vessel instance, and *context* is the pointer passed to CreateAirfoil3. It can be used to make available to the callback function any additional parameters required to calculate the lift and drag coefficients. All other parameters are identical to AirfoilCoeffFunc (see [CreateAirfoil](#)).

**See also:**

[CreateAirfoil](#), [CreateAirfoil2](#), [EditAirfoil](#), [DelAirfoil](#)

**16.51.3.85 bool VESSEL::GetAirfoilParam (AIRFOILHANDLE *hAirfoil*, VECTOR3 \* *ref*, AirfoilCoeffFunc \* *cf*, void \*\* *context*, double \* *c*, double \* *S*, double \* *A*) const**

Returns the parameters of an existing airfoil.

**Parameters:**

- ← *hAirfoil* airfoil handle
- *ref* pointer to centre of pressure [m]
- *cf* pointer to aerodynamic coefficient callback function
- *c* pointer to chord length [m]
- *S* pointer to wing area [ $m^2$  ]
- *A* pointer to wing aspect ratio
- *context* pointer to callback context data

**Returns:**

*false* indicates failure

**Note:**

This function copies the airfoil parameters into the variables referenced by the pointers in the parameter list.

Any pointers set to NULL are ignored.

```
VECTOR3 cop;
AirfoilCoeffFunc cf;
void *context;
double c, S, A;
v->GetAirfoilParam(hAirfoil, &cop, &cf, &context, &c, &S, &A);
```

**See also:**

[CreateAirfoil](#), [CreateAirfoil2](#), [CreateAirfoil3](#), [EditAirfoil](#), [DelAirfoil](#)

**16.51.3.86 void VESSEL::EditAirfoil (AIRFOILHANDLE *hAirfoil*, DWORD *flag*, const VECTOR3 & *ref*, AirfoilCoeffFunc *cf*, double *c*, double *S*, double *A*) const**

Resets the parameters of an existing airfoil definition.

**Parameters:**

- hAirfoil* airfoil handle
- flag* bitflags to define which parameters to reset (see notes)
- ref* new centre of pressure
- cf* new callback function for coefficient calculation
- c* new chord length [m]
- S* new wing area [ $m^2$  ]
- A* new wing aspect ratio

**Note:**

This function can be used to modify the parameters of a previously created airfoil.

*flag* contains the bit flags defining which parameters will be modified. It can be any combination of the following:

0x01	modify force attack point
0x02	modify coefficient callback function
0x04	modify chord length
0x08	modify wing area
0x10	modify wing aspect ratio

If the airfoil identified by *hAirfoil* was created with [CreateAirfoil3](#), and you want to modify the callback function, then *cf* must point to a function with *AirfoilCoeffFuncEx* interface, and must be cast to *AirfoilCoeffFunc* when passed to *EditAirfoil*.

**See also:**

[CreateAirfoil2](#), [CreateAirfoil3](#)

**16.51.3.87 bool VESSEL::DelAirfoil (AIRFOILHANDLE *hAirfoil*) const**

Deletes a previously defined airfoil.

**Parameters:**

*hAirfoil* airfoil handle

**Returns:**

*false* indicates failure (invalid handle)

**Note:**

If all the vessel's airfoils are deleted without creating new ones, Orbiter reverts to the obsolete legacy atmospheric flight model.

**See also:**

[CreateAirfoil2](#), [CreateAirfoil3](#), [ClearAirfoilDefinitions](#)

**16.51.3.88 void VESSEL::ClearAirfoilDefinitions () const**

Removes all airfoils currently defined for the vessel.

**Note:**

This function is useful if a vessel needs to re-define all its airfoil definitions as a result of a structural change.

After clearing all airfoils, you should generate new ones. Even wingless objects (such as capsules) should define their aerodynamic behaviour by airfoils (see [CreateAirfoil](#)). Vessels without airfoil definitions revert to the obsolete legacy atmospheric flight model.

**See also:**

[DelAirfoil](#), [CreateAirfoil](#), [CreateAirfoil2](#), [CreateAirfoil3](#)

**16.51.3.89 void VESSEL::CreateControlSurface (AIRCTRL\_TYPE *type*, double *area*, double *dCl*, const VECTOR3 & *ref*, int *axis* = AIRCTRL\_AXIS\_AUTO, UINT *anim* = (UINT)-1) const**

Creates an aerodynamic control surface.

**Parameters:**

*type* control surface type (see [Aerodynamic control surface types](#))

*area* control surface area [ $\text{m}^2$  ]

*dCl* shift in lift coefficient achieved by fully extended control

*ref* centre of pressure in vessel coordinates [m]

*axis* rotation axis (see [Control surface axis orientation](#))

*anim* animation reference, if applicable

**Note:**

Control surfaces include elevators, rudders, ailerons, flaps, etc. They can be used to control the vessel during atmospheric flight.

When selecting automatic axis control (axis=AIRCTRL\_AXIS\_AUTO), the following axes will be used for given control surfaces:

Elevator	XPOS
Rudder	YPOS
Aileron	XPOS if ref.x > 0, XNEG otherwise
Flap	XPOS

For ailerons, at least 2 control surfaces should be defined (e.g. on left and right wing) with opposite rotation axes, to obtain the angular momentum for banking the vessel.

Elevators typically use the XPOS axis, assuming the that the centre of pressure is aft of the centre of gravity. If pitch control is provided by a canard configuration *ahead* of the CoG, XNEG should be used instead.

The centre of pressure defined by the *ref* parameter is the point at which the lift and drag forces for the control surface are applied.

To improve performance, multiple control surfaces may sometimes be defined by a single call to CreateControlSurface. For example, the elevator controls on the left and right wing may be combined by setting a centered attack point.

Control surfaces can be animated, by passing an animation reference to CreateControlSurface. The animation reference is obtained when creating the animation with [CreateAnimation](#). The animation should support a state in the range from 0 to 1, with neutral surface position at state 0.5.

**See also:**

[CreateControlSurface2](#), [CreateControlSurface3](#)

**16.51.3.90 CTRLSURFHANDLE VESSEL::CreateControlSurface2 (AIRCTRL\_TYPE *type*, double *area*, double *dCl*, const VECTOR3 & *ref*, int *axis* = AIRCTRL\_AXIS\_AUTO, UINT *anim* = (UINT)-1) const**

Creates an aerodynamic control surface and returns a handle.

**Parameters:**

*type* control surface type (see [Aerodynamic control surface types](#))

*area* control surface area [ $\text{m}^2$  ]

*dCl* shift in lift coefficient achieved by fully extended control

*ref* centre of pressure in vessel coordinates [m]  
*axis* rotation axis (see [Control surface axis orientation](#))  
*anim* animation reference, if applicable

**Returns:**

Control surface handle.

**Note:**

This function is identical to [CreateControlSurface](#), but it returns a handle for later reference (e.g. to delete it with [DelControlSurface](#))  
It is equivalent to [CreateControlSurface3](#) with *delay* = 1.

**See also:**

[CreateControlSurface](#), [CreateControlSurface3](#), [DelControlSurface](#)

**16.51.3.91 CTRLSURFHANDLE VESSEL::CreateControlSurface3 (AIRCTRL\_TYPE *type*, double *area*, double *dCl*, const VECTOR3 & *ref*, int *axis* = AIRCTRL\_AXIS\_AUTO, double *delay* = 1.0, UINT *anim* = (UINT)-1) const**

Creates an aerodynamic control surface and returns a handle.

**Parameters:**

*type* control surface type (see [Aerodynamic control surface types](#))  
*area* control surface area [ $m^2$ ]  
*dCl* shift in lift coefficient achieved by fully extended control  
*ref* centre of pressure in vessel coordinates [m]  
*axis* rotation axis (see [Control surface axis orientation](#))  
*delay* response delay setting [s]  
*anim* animation reference, if applicable

**Returns:**

Control surface handle.

**Note:**

This function is identical to [CreateControlSurface2](#) except that it specifies an additional 'delay' parameter which defines the response delay for the surface (the time it takes to move from neutral to fully deployed). Setting delay=0 provides direct response.

**See also:**

[CreateControlSurface](#), [CreateControlSurface2](#)

**16.51.3.92 bool VESSEL::DelControlSurface (CTRLSURFHANDLE *hCtrlSurf*) const**

Deletes a previously defined aerodynamic control surface.

**Parameters:**

*hCtrlSurf* control surface handle

**Returns:**

*false* indicates error (invalid handle)

**See also:**

[CreateControlSurface2](#), [CreateControlSurface3](#)

**16.51.3.93 void VESSEL::ClearControlSurfaceDefinitions () const**

Removes all aerodynamic control surfaces.

**Note:**

This function is useful if the vessel has to re-define all its control surfaces (e.g. as a result of structural change).

**See also:**

[DelControlSurface](#)

**16.51.3.94 void VESSEL::SetControlSurfaceLevel (AIRCTRL\_TYPE *type*, double *level*) const**

Updates the position of an aerodynamic control surface.

**Parameters:**

*type* control surface type (see [Aerodynamic control surface types](#))

*level* new control surface position [-1...+1]

**Note:**

Parameter *level* defines a *target* state for the surface. Control surfaces generally require a finite amount of time to move from the current to the target state.

This method affects the *permanent* setting of the control surface, while manual input via keyboard or joystick affects the *transient* setting. The total target state of the control surface is the sum of both settings, clamped to the range [-1...+1]

**See also:**

[SetControlSurfaceLevel\(AIRCTRL\\_TYPE,double,bool\)const](#), [GetControlSurfaceLevel](#)

**16.51.3.95 void VESSEL::SetControlSurfaceLevel (AIRCTRL\_TYPE *type*, double *level*, bool *direct*) const**

Updates the position of an aerodynamic control surface.

**Parameters:**

*type* control surface type (see [Aerodynamic control surface types](#))

*level* new control surface position [-1...+1]

*direct* application mode

**Note:**

If parameter *direct==true* then the specified level is applied directly, bypassing any reaction delays defined for the control surface.

If parameter *direct==false* then this method is equivalent to [SetControlSurfaceLevel\(AIRCTRL\\_TYPE,double\)const](#).

Bypassing the response delay can be useful for debugging autopilots etc. but should be avoided in production code, since it is unphysical. If you want to simulate fast-responding controls, create the surface with a small delay setting instead.

**See also:**

[SetControlSurfaceLevel\(AIRCTRL\\_TYPE,double\)const](#), [CreateControlSurface3](#)

**16.51.3.96 double VESSEL::GetControlSurfaceLevel (AIRCTRL\_TYPE *type*) const**

Returns the current position of a control surface.

**Parameters:**

*type* control surface type (see [Aerodynamic control surface types](#))

**Returns:**

Current position of the surface [-1..+1]

**Note:**

This method returns the *actual*, not the *target* position. Due to finite response time, it may therefore not return the value set by a preceding call to [SetControlSurfaceLevel](#).

**See also:**

[SetControlSurfaceLevel](#)

**16.51.3.97 void VESSEL::CreateVariableDragElement (double \* *drag*, double *factor*, const VECTOR3 & *ref*) const**

Attaches a modifiable drag component to the vessel.

**Parameters:**

*drag* pointer to external drag control parameter

*factor* drag magnitude scaling factor

*ref* drag force attack point [m]

**Note:**

This method is useful for defining drag produced by movable parts such as landing gear and airbrakes. The magnitude of the drag force is calculated as

$$D = d \cdot f \cdot q_{\infty}$$

where  $d$  is the control parameter,  $f$  is the scale factor, and  $q$  is the freestream dynamic pressure. The value of  $d$  (the parameter pointed to by *drag*) should be set to values between 0 (no drag) and 1 (full drag). Any changes to the value have immediate effect.

Depending on the attack point, the applied drag force may create torque in addition to linear force.

**See also:**

[ClearVariableDragElements](#)

**16.51.3.98 void VESSEL::ClearVariableDragElements () const**

Removes all drag elements defined with CreateVariableDragElement.

**See also:**

[CreateVariableDragElement](#)

**16.51.3.99 void VESSEL::GetCW (double & cw\_z\_pos, double & cw\_z\_neg, double & cw\_x, double & cw\_y) const**

Returns the vessel's wind resistance coefficients (legacy flight model only).

**Parameters:**

*cw\_z\_pos* resistance coefficient in positive z direction (forward)

*cw\_z\_neg* resistance coefficient in negative z direction (back)

*cw\_x* resistance coefficient in lateral direction (left/right)

*cw\_y* resistance coefficient in vertical direction (up/down)

**Note:****[Legacy aerodynamic flight model only]**

The cw coefficients are only used by the legacy flight model (if no airfoils are defined). In the presence of airfoils, drag calculations are performed on the basis of the airfoil parameters.

The first value (*cw\_z\_pos*) is the coefficient used if the vessel's airspeed z-component is positive (vessel moving forward). The second value is used if the z-component is negative (vessel moving backward).

Lateral and vertical components are assumed symmetric.

**See also:**

[SetCW](#), [CreateAirfoil](#)

**16.51.3.100 void VESSEL::SetCW (double *cw\_z\_pos*, double *cw\_z\_neg*, double *cw\_x*, double *cw\_y*) const**

Set the vessel's wind resistance coefficients along its axis directions.

**Parameters:**

- cw\_z\_pos* coefficient in positive z direction (forward)
- cw\_z\_neg* coefficient in negative z direction (back)
- cw\_x* coefficient in lateral direction (left/right)
- cw\_y* coefficient in vertical direction (up/down)

**Note:****[Legacy aerodynamic flight model only]**

The cw coefficients are only used by the legacy flight model (if no airfoils are defined). In the presence of airfoils, drag calculations are performed on the basis of the airfoil parameters.

The first value (*cw\_z\_pos*) is the coefficient used if the vessel's airspeed z-component is positive (vessel moving forward). The second value is used if the z-component is negative (vessel moving backward).

Lateral and vertical components are assumed symmetric.

**See also:**

[GetCW](#), [CreateAirfoil](#)

**16.51.3.101 double VESSEL::GetWingAspect () const**

Returns the vessel's wing aspect ratio ( $\text{wingspan}^2 / \text{wing area}$ ).

**Returns:**

Wing aspect ratio ( $\text{wingspan}^2 / \text{wing area}$ )

**Note:****[Legacy aerodynamic flight model only]**

The aspect ratio returned by this function is only used by the legacy aerodynamic flight model. If the vessel uses the new flight model (i.e. defines at least one airfoil), then this value is ignored, and the airfoil parameters are used instead.

The aspect ratio is used in the calculation of induced drag.

**See also:**

[SetWingAspect](#), [GetWingEffectiveness](#), [CreateAirfoil](#)

**16.51.3.102 void VESSEL::SetWingAspect (double *aspect*) const**

Set the wing aspect ratio ( $\text{wingspan}^2 / \text{wing area}$ ).

**Parameters:**

- aspect* wing aspect ratio

**Note:****[Legacy aerodynamic flight model only]**

This function defines the wing aspect ratio for the legacy flight model. If the vessel uses the new flight model (i.e. defines at least one airfoil), then this value is ignored, and the airfoil parameters are used instead.

The aspect ratio is used in the calculation of induced drag.

**See also:**

[GetWingAspect](#), [SetWingEffectiveness](#), [CreateAirfoil](#)

**16.51.3.103 double VESSEL::GetWingEffectiveness () const**

Returns the wing form factor used in aerodynamic calculations.

**Returns:**

wing form factor

**Note:****[Legacy aerodynamic flight model only]**

The form factor returned by this function is only used by the legacy aerodynamic flight model. If the vessel uses the new flight model (i.e. defines at least one airfoil), then this value is ignored, and the airfoil parameters are used instead.

The form factor, together with the aspect ratio, determines the amount of induced drag for given lift. Higher values of the form factor result in lower drag.

Typical values are  $\sim 3.1$  for elliptic wings,  $\sim 2.8$  for tapered wings, and  $\sim 2.5$  for rectangular wings. Default is 2.8.

**See also:**

[SetWingEffectiveness](#), [GetWingAspect](#), [CreateAirfoil](#)

**16.51.3.104 void VESSEL::SetWingEffectiveness (double *eff*) const**

Set the wing form factor for aerodynamic lift and drag calculations.

**Parameters:**

*eff* wing form factor

**Note:****[Legacy aerodynamic flight model only]**

This function defines the wing form factor for the legacy flight model. If the vessel uses the new flight model (i.e. defines at least one airfoil), then this value is ignored, and the airfoil parameters are used instead.

The form factor, together with the aspect ratio, determines the amount of induced drag for given lift. Higher values of the form factor result in lower drag.

Typical values for *eff* are:  $\sim 3.1$  for elliptic wings,  $\sim 2.8$  for tapered wings,  $\sim 2.5$  for rectangular wings.

**See also:**

[GetWingEffectiveness](#), [SetWingAspect](#), [CreateAirfoil](#)

**16.51.3.105 void VESSEL::GetRotDrag (VECTOR3 & rd) const**

Returns the vessel's atmospheric rotation resistance coefficients.

**Parameters:**

*rd* drag coefficients for rotation around the 3 vessel axes

**Note:**

*rd* contains the components  $r_{x,y,z}$  against rotation around the local vessel axes in atmosphere, where angular deceleration due to atmospheric friction is defined as  $a_{x,y,z} = -w_{x,y,z} q S_y r_{x,y,z}$  with angular velocity *w*, dynamic pressure *q* and reference surface *Sy*, defined by the vessel's cross section projected along the vertical (*y*) axis.

**See also:**

[SetRotDrag](#)

**16.51.3.106 void VESSEL::SetRotDrag (const VECTOR3 & rd) const**

Set the vessel's atmospheric rotation resistance coefficients.

**Parameters:**

*rd* drag coefficients for rotation around the 3 vessel axes

**Note:**

*rd* contains the components  $r_{x,y,z}$  against rotation around the local vessel axes in atmosphere, where angular deceleration due to atmospheric friction is defined as  $a_{x,y,z} = -w_{x,y,z} q S_y r_{x,y,z}$  with angular velocity *w*, dynamic pressure *q* and reference surface *Sy*, defined by the vessel's cross section projected along the vertical (*y*) axis.

**See also:**

[GetRotDrag](#)

**16.51.3.107 double VESSEL::GetPitchMomentScale () const**

Returns the scaling factor for the pitch moment.

**Returns:**

pitch moment scale factor

**Note:**

The pitch moment is the angular moment around the vessel's lateral (*x*) axis occurring in atmospheric flight. It works toward reducing the pitch angle (angle of attack).

The larger the scaling factor, the stronger the effect becomes ("stiff handling")

This value is only used with the old aerodynamic flight model, i.e. if no airfoils have been defined.

**See also:**

[SetPitchMomentScale](#), [GetYawMomentScale](#), [CreateAirfoil](#)

**16.51.3.108 void VESSEL::SetPitchMomentScale (double *scale*) const**

Sets the scaling factor for the pitch moment.

**Parameters:**

*scale* scale factor for pitch moment

**Note:**

The pitch moment is the angular moment around the vessel's lateral (x) axis occurring in atmospheric flight. It works toward reducing the pitch angle (angle of attack) between the vessel's longitudinal axis and the airstream vector.

The larger the scaling factor, the stronger the effect becomes ("stiff handling")

This value is only used with the old aerodynamic flight model, i.e. if not airfoils have been defined.

The default value is 0.

**See also:**

[GetPitchMomentScale](#), [GetYawMomentScale](#), [CreateAirfoil](#)

**16.51.3.109 double VESSEL::GetYawMomentScale () const**

Returns the scaling factor for the yaw moment.

**Returns:**

yaw moment scale factor

**Note:**

The yaw moment is the angular moment around the vessel's vertical (y) axis occurring in atmospheric flight. It works toward reducing the slip angle between the vessel's longitudinal axis and the airstream vector.

This value is only used with the old aerodynamic flight model, i.e. if no airfoils have been defined.

**See also:**

[SetYawMomentScale](#), [GetPitchMomentScale](#), [CreateAirfoil](#)

**16.51.3.110 void VESSEL::SetYawMomentScale (double *scale*) const**

Sets the scaling factor for the yaw moment.

**Parameters:**

*scale* scale factor for yaw angle moment.

**Note:**

The yaw moment is the angular moment around the vessel's vertical (y) axis occurring in atmospheric flight. It works toward reducing the slip angle between the vessel's longitudinal axis and the airstream vector.

This value is only used with the old aerodynamic flight model, i.e. if not airfoils have been defined.

The default value is 0.

**See also:**

[SetPitchMomentScale](#), [GetYawMomentScale](#), [CreateAirfoil](#)

**16.51.3.111 double VESSEL::GetTrimScale () const**

Returns the scaling factor for the pitch trim control.

**Returns:**

pitch trim scale factor.

**Note:**

This function returns the value previously set with SetTrimScale  
It is only used with the old atmospheric flight model (if no airfoils have been defined).

**See also:**

[SetTrimScale](#), [GetPitchMomentScale](#), [GetYawMomentScale](#), [CreateAirfoil](#)

**16.51.3.112 void VESSEL::SetTrimScale (double *scale*) const**

Sets the scaling factor for the pitch trim control.

**Parameters:**

*scale* pitch trim scaling factor

**Note:**

This method is used only in combination with the old flight model, that is, if the vessel doesn't define any airfoils. In the new flight model, this has been replaced by CreateControlSurface (AIRCTRL\_ELEVATORTRIM, ...).

If scale is set to zero (default) the vessel does not have a pitch trim control.

**See also:**

[GetTrimScale](#), [SetPitchMomentScale](#), [GetYawMomentScale](#), [CreateAirfoil](#), [CreateControlSurface](#)

**16.51.3.113 void VESSEL::SetLiftCoeffFunc (LiftCoeffFunc *lcf*) const**

Defines the callback function for aerodynamic lift calculation.

**Parameters:**

*lcf* pointer to callback function (see notes)

**Note:****[Legacy aerodynamic flight model only]**

This method defines callback function for lift calculation as a function of angle of attack for the legacy flight model. If the vessel uses the new flight model (i.e. defines at least one airfoil), then this value is ignored, and the airfoil parameters are used instead.

The interface of the callback function is defined as

```
typedef double (*LiftCoeffFunc) (double aoa)
```

where  $aoa$  is the angle of attack [rad], and the return value is the resulting lift coefficient. The callback function must be able to process input  $aoa$  values in the range  $-Pi \dots +Pi$ . The preferred method for defining lift and drag characteristics is via the CreateAirfoil method, which is much more versatile. Orbiter ignores the SetLiftCoeffFunc function if any airfoils have been created. If neither airfoils are defined, nor this method is called, then the default behaviour is not to generate any aerodynamic lift.

**See also:**

[CreateAirfoil](#)

**16.51.3.114 double VESSEL::GetLift () const**

Returns magnitude of aerodynamic lift force vector.

**Returns:**

Magnitude of lift force vector [N].

**Note:**

Return value is the sum of lift components from all airfoils.

**See also:**

[GetLiftVector](#), [GetDrag](#)

**16.51.3.115 double VESSEL::GetDrag () const**

Returns magnitude of aerodynamic drag force vector.

**Returns:**

Magnitude of drag force vector [N].

**Note:**

Return value is the sum of drag components from all airfoils.

**See also:**

[GetDragVector](#), [GetLift](#)

**16.51.3.116 bool VESSEL::GetWeightVector (VECTOR3 & G) const**

Returns gravitational force vector in local vessel coordinates.

**Parameters:**

→  $G$  gravitational force vector [N]

**Returns:**

Always true.

**Note:**

When the vessel status is updated dynamically, G is composed of all gravity sources currently used for the vessel propagation (excluding sources with contributions below threshold).

During orbit stabilisation, only the contribution from the primary source is returned.

**See also:**

[GetThrustVector](#), [GetLiftVector](#), [GetDragVector](#), [GetForceVector](#)

**16.51.3.117 bool VESSEL::GetThrustVector (VECTOR3 & T) const**

Returns thrust force vector in local vessel coordinates.

**Parameters:**

→  $T$  thrust vector [N]

**Returns:**

false indicates zero thrust. In that case, the returned vector is (0,0,0).

**Note:**

On return, T contains the vector sum of thrust components from all engines.

This function provides information about the linear thrust force, but not about the angular moment (torque) induced.

**See also:**

[GetWeightVector](#), [GetLiftVector](#), [GetDragVector](#), [GetForceVector](#)

**16.51.3.118 bool VESSEL::GetLiftVector (VECTOR3 & L) const**

Returns aerodynamic lift force vector in local vessel coordinates.

**Parameters:**

→  $L$  lift vector [N]

**Returns:**

false indicates zero lift. In that case, the returned vector is (0,0,0).

**Note:**

Return value is the sum of lift components from all airfoils.

The lift vector is perpendicular to the relative wind (and thus to the drag vector) and has zero x-component.

**See also:**

[GetLift](#), [GetWeightVector](#), [GetThrustVector](#), [GetDragVector](#), [GetForceVector](#)

**16.51.3.119 bool VESSEL::GetDragVector (VECTOR3 & D) const**

Returns aerodynamic drag force vector in local vessel coordinates.

**Parameters:**

→  $D$  drag vector [N]

**Returns:**

false indicates zero drag. In that case, the returned vector is (0,0,0).

**Note:**

On return, D contains the sum of drag components from all airfoils.  
The drag vector is parallel to the relative wind (direction of air flow).

**See also:**

[GetDrag](#), [GetWeightVector](#), [GetThrustVector](#), [GetLiftVector](#), [GetForceVector](#)

**16.51.3.120 bool VESSEL::GetForceVector (VECTOR3 & F) const**

Returns total force vector acting on the vessel in local vessel coordinates.

**Parameters:**

→  $F$  total force vector [N]

**Returns:**

Always true

**Note:**

On return, F contains the sum of all forces acting on the vessel.  
This may not be equal to the sum of weight, thrust, lift and drag vectors, because it also includes surface contact forces, user-defined forces and any other forces.

**See also:**

[GetWeightVector](#), [GetThrustVector](#), [GetLiftVector](#), [GetDragVector](#), [GetTorqueVector](#)

**16.51.3.121 bool VESSEL::GetTorqueVector (VECTOR3 & M) const**

Returns the total torque vector acting on the vessel in local vessel coordinates.

**Parameters:**

→  $M$  total torque vector [Nm]

**Returns:**

Always true

**Note:**

On return, M contains the total torque vector acting on the vessel in its centre of mass. The torque vector contains contributions from thrusters, aerodynamic forces and gravity gradient effects (if enabled).

**See also:**[GetForceVector](#)**16.51.3.122 void VESSEL::AddForce (const VECTOR3 & F, const VECTOR3 & r) const**

Add a custom body force.

**Parameters:**

*F* force vector [N]

*r* force attack point in local vessel coordinates [m]

**Note:**

This function can be used to implement custom forces (braking chutes, tethers, etc.). It should not be used for standard forces such as engine thrust or aerodynamic forces which are handled internally (although in theory this function makes it possible to bypass Orbiter's built-in thrust and aerodynamics model completely and replace it by a user-defined model).

The force is applied only for the next time step. AddForce will therefore usually be used inside the [VESSEL2::clbkPreStep](#) callback function.

**See also:**[GetForceVector](#)**16.51.3.123 PROPELLANT\_HANDLE VESSEL::CreatePropellantResource (double maxmass, double mass = -1.0, double efficiency = 1.0) const**

Create a new propellant resource ("fuel tank").

Propellant resources are a component of the vessel's propulsion system. They can hold propellants and distribute them to connected engines to generate thrust.

**Parameters:**

*maxmass* maximum propellant capacity of the tank [kg]

*mass* initial propellant mass of the resource [kg]

*efficiency* fuel efficiency factor (>0)

**Returns:**

propellant resource handle

**Note:**

Orbiter doesn't distinguish between propellant and oxidant. A "propellant resource" is assumed to be a combination of fuel and oxidant resources.

The interpretation of a propellant resource (liquid or solid propulsion system, ion drive, etc.) is up to the vessel developer.

The rate of fuel consumption depends on the thrust level and Isp (fuel-specific impulse) of the thrusters attached to the resource.

The fuel efficiency rating, together with a thruster's Isp rating, determines how much fuel is consumed per second to obtain a given thrust:  $R = F(e \cdot Isp)^{-1}$  with fuel rate R [kg/s], thrust F [N], efficiency e and fuel-specific impulse Isp [m/s].

If mass < 0 then mass = maxmass is substituted.

**See also:**

[DelPropellantResource](#), [SetPropellantMaxMass](#), [SetPropellantMass](#), [SetPropellantEfficiency](#), [GetPropellantMaxMass](#), [GetPropellantMass](#), [GetPropellantEfficiency](#)

**16.51.3.124 void VESSEL::DelPropellantResource (PROPELLANT\_HANDLE & *ph*) const**

Remove a propellant resource.

**Parameters:**

*ph* propellant resource handle (NULL on return)

**Note:**

If any thrusters were attached to this fuel resource, they are disabled until connected to a new fuel resource.

**See also:**

[CreatePropellantResource](#), [ClearPropellantResources](#)

**16.51.3.125 void VESSEL::ClearPropellantResources () const**

Remove all propellant resources for the vessel.

**Note:**

After a call to this function, all the vessel's thrusters will be disabled until they are linked to new resources.

**See also:**

[DelPropellantResource](#)

**16.51.3.126 DWORD VESSEL::GetPropellantCount () const**

Returns the current number of vessel propellant resources.

**Returns:**

Number of propellant resources currently defined for the vessel.

**See also:**

[CreatePropellantResource](#), [GetPropellantHandleByIndex](#)

**16.51.3.127 PROPELLANT\_HANDLE VESSEL::GetPropellantHandleByIndex (DWORD *idx*) const**

Returns the handle of a propellant resource for a given index.

**Parameters:**

*idx* propellant resource index ( $\geq 0$ )

**Returns:**

Propellant resource handle

**Note:**

The index must be in the range between 0 and [GetPropellantCount\(\)](#)-1. If the index is out of range, the returned handle is NULL.

The index of a given propellant resource may change if any resources are deleted. The handle remains valid until the corresponding resource is deleted.

**See also:**

[CreatePropellantResource](#), [GetPropellantCount](#)

**16.51.3.128 double VESSEL::GetPropellantMaxMass (PROPELLANT\_HANDLE *ph*) const**

Returns the maximum capacity of a propellant resource.

**Parameters:**

*ph* propellant resource handle

**Returns:**

Max. propellant capacity [kg].

**See also:**

[SetPropellantMaxMass](#), [GetPropellantMass](#), [SetPropellantMass](#)

**16.51.3.129 void VESSEL::SetPropellantMaxMass (PROPELLANT\_HANDLE *ph*, double *maxmass*) const**

Reset the maximum capacity of a fuel resource.

**Parameters:**

*ph* propellant resource handle

*maxmass* max. fuel capacity ( $\geq 0$ ) [kg]

**Note:**

The actual fuel mass contained in the tank is not affected by this function, unless the new maximum propellant mass is less than the current fuel mass, in which case the fuel mass is reduced to the maximum capacity.

**See also:**

[GetPropellantMaxMass](#), [SetPropellantMass](#), [GetPropellantMass](#), [GetTotalPropellantMass](#), [GetFuel-Mass](#), [SetPropellantEfficiency](#)

**16.51.3.130 double VESSEL::GetPropellantMass (PROPELLANT\_HANDLE *ph*) const**

Returns the current mass of a propellant resource.

**Parameters:**

*ph* propellant resource handle

**Returns:**

Current propellant mass [kg].

**See also:**

[SetPropellantMass](#), [GetPropellantMaxMass](#), [SetPropellantMaxMass](#)

**16.51.3.131 void VESSEL::SetPropellantMass (PROPELLANT\_HANDLE *ph*, double *mass*) const**

Reset the current mass of a propellant resource.

**Parameters:**

*ph* propellant resource handle

*mass* propellant mass ( $\geq 0$ ) [kg]

**Note:**

$0 \leq \text{mass} \leq \text{maxmass}$  is required, where maxmass is the maximum capacity of the propellant resource.

This method should be used to simulate refuelling, fuel leaks, cross-feeding between tanks, etc. but not for normal fuel consumption by thrusters (which is handled internally by the Orbiter core).

**See also:**

[GetPropellantMass](#), [SetPropellantMaxMass](#), [GetTotalPropellantMass](#), [GetFuelMass](#), [SetPropellantEfficiency](#)

**16.51.3.132 double VESSEL::GetTotalPropellantMass () const**

Returns the vessel's current total propellant mass.

**Returns:**

Sum of current mass of all propellant resources defined for the vessel [kg].

**See also:**

[GetPropellantMass](#), [GetPropellantMaxMass](#)

**16.51.3.133 double VESSEL::GetPropellantEfficiency (PROPELLANT\_HANDLE *ph*) const**

Returns the efficiency factor of a propellant resource.

**Parameters:**

*ph* propellant resource handle

**Returns:**

Fuel efficiency factor

**Note:**

The fuel efficiency rating, together with a thruster's Isp rating, determines how much fuel is consumed per second to obtain a given thrust:  $R = F/(e \text{ Isp})$  with fuel rate R [kg/s], thrust F [N], efficiency e and fuel-specific impulse Isp [m/s].

**See also:**

[SetPropellantEfficiency](#), [GetPropellantMaxMass](#), [CreatePropellantResource](#)

**16.51.3.134 void VESSEL::SetPropellantEfficiency (PROPELLANT\_HANDLE *ph*, double *efficiency*) const**

Reset the efficiency factor of a fuel resource.

**Parameters:**

*ph* propellant resource handle

*efficiency* fuel efficiency factor ( $> 0$ )

**Note:**

The fuel efficiency rating, together with a thruster's Isp rating, determines how much fuel is consumed per second to obtain a given thrust:  $R = F/(e \text{ Isp})$  with fuel rate R [kg/s], thrust F [N], efficiency e and fuel-specific impulse Isp [m/s].

**See also:**

[GetPropellantEfficiency](#), [CreatePropellantResource](#), [SetPropellantMaxMass](#), [SetPropellantMass](#)

**16.51.3.135 double VESSEL::GetPropellantFlowrate (PROPELLANT\_HANDLE *ph*) const**

Returns the current mass flow rate from a propellant resource.

**Parameters:**

*ph* propellant resource handle

**Returns:**

Current propellant mass flow rate [kg/s].

**See also:**

[GetPropellantMass](#), [GetTotalPropellantFlowrate](#), [GetFuelRate](#)

**16.51.3.136 double VESSEL::GetTotalPropellantFlowrate () const**

Returns the current total mass flow rate, summed over all propellant resources.

**Returns:**

Total propellant mass flow rate [kg/s].

**See also:**

[GetPropellantFlowrate](#), [GetFuelRate](#)

**16.51.3.137 void VESSEL::SetDefaultPropellantResource (PROPELLANT\_HANDLE *ph*) const**

Define a "default" propellant resource.

This is used for the various legacy fuel-related API functions, and for the "Fuel" indicator in the generic panel-less HUD display.

**Parameters:**

*ph* propellant resource handle

**Note:**

If this function is not called, the first propellant resource is used as default.

**See also:**

[CreatePropellantResource](#), [GetDefaultPropellantResource](#)

**16.51.3.138 PROPELLANT\_HANDLE VESSEL::GetDefaultPropellantResource () const**

Returns the handle for the vessel's default propellant resource.

**Returns:**

Default propellant resource handle

**See also:**

[SetDefaultPropellantResource](#)

**16.51.3.139 double VESSEL::GetMaxFuelMass () const**

Returns the maximum capacity of the vessel's default propellant resource.

**Returns:**

Max. capacity of default propellant resource [kg].

**Note:**

The function returns 0 if no fuel resources are defined.

**See also:**

[GetPropellantMaxMass](#), [SetDefaultPropellantResource](#)

**16.51.3.140 void VESSEL::SetMaxFuelMass (double *mass*) const**

Set the maximum fuel capacity of the vessel's default propellant resource.

**Parameters:**

*mass* max. propellant mass [kg].

**Note:**

If no propellant resources are defined for the vessel, a call to this method creates a new propellant resource with the specified capacity.

If the vessel already contains propellant resources, this method resets the maximum capacity of the vessel's default resource.

**See also:**

[SetPropellantMaxMass](#), [SetDefaultPropellantResource](#)

**16.51.3.141 double VESSEL::GetFuelMass () const**

Returns the current mass of the vessel's default propellant resource.

**Returns:**

Current mass of the default propellant resource [kg].

**See also:**

[GetPropellantMass](#), [SetDefaultPropellantResource](#)

**16.51.3.142 void VESSEL::SetFuelMass (double *mass*) const**

Reset the current mass of the vessel's default propellant resource.

**Parameters:**

*mass* new propellant mass [kg].

**Note:**

*mass* must be between 0 and the maximum capacity of the propellant resource.

If the vessel has not defined any propellant resources, this method has no effect.

**See also:**

[GetFuelMass](#), [SetPropellantMass](#), [SetMaxFuelMass](#), [SetDefaultPropellantResource](#)

**16.51.3.143 double VESSEL::GetFuelRate () const**

Returns the current mass flow rate from the default propellant resource.

**Returns:**

Current mass flow rate from the default propellant resource [kg/s].

**See also:**

[GetPropellantFlowrate](#), [SetDefaultPropellantResource](#)

**16.51.3.144 THRUSTER\_HANDLE VESSEL::CreateThruster (const VECTOR3 & *pos*, const VECTOR3 & *dir*, double *maxth0*, PROPELLANT\_HANDLE *hp* = NULL, double *isp0* = 0 . 0, double *isp\_ref* = 0 . 0, double *p\_ref* = 101 . 4e3) const**

Add a logical thruster definition for the vessel.

**Parameters:**

*pos* thrust force attack point in vessel coordinates [m]  
*dir* thrust force direction in vessel coordinates (normalised)  
*maxth0* max. vacuum thrust rating [N]  
*hp* propellant resource feeding the thruster  
*isp0* vacuum fuel-specific impulse (Isp) [m/s]  
*isp\_ref* Isp value at reference pressure *p\_ref* [m/s]  
*p\_ref* reference pressure for Isp\_ref [Pa]

**Returns:**

Thruster identifier

**Note:**

The fuel-specific impulse defines how much thrust is produced by burning 1kg of fuel per second. If the Isp level is not specified or is = 0, a default value is used (see [SetISP\(\)](#)).

To define the thrust and Isp ratings to be pressure-dependent, specify an *isp\_ref* value > 0, and set *p\_ref* to the corresponding atmospheric pressure. Thrust and Isp at pressure *p* will then be calculated as

$$\text{Isp}(p) = \text{Isp}_0(1 - p\eta), \quad \text{Th}(p) = \text{Th}_0(1 - p\eta), \quad \text{where} \quad \eta = \frac{\text{Isp}_0 - \text{Isp}_{\text{ref}}}{p_{\text{ref}} \text{Isp}_0}$$

If *isp\_ref* = 0 then no pressure-dependency is assumed ( $\eta = 0$ ).

If no propellant resource is specified, the thruster is disabled until it is linked to a resource by [SetThrusterResource\(\)](#).

If *isp0* <= 0, then the default Isp value is substituted (see [SetISP\(\)](#)).

Thruster forces can create linear as well as angular moments, depending on the attack point and direction.

Use CreateThrusterGroup to assemble thrusters into logical groups.

**See also:**

[DelThruster](#), [CreateThrusterGroup](#), [AddExhaust](#), [SetISP](#), [SetThrusterIsp](#), [SetThrusterResource](#)

**16.51.3.145 bool VESSEL::DelThruster (THRUSTER\_HANDLE & *th*) const**

Delete a logical thruster definition.

**Parameters:**

*th* thruster handle (NULL on return)

**Returns:**

true on success, false if the supplied thruster handle was invalid.

**Note:**

Deleted thrusters will be automatically removed from all thruster groups they had been assigned to.  
All exhaust render definitions which refer to the deleted thruster are removed.

**See also:**

[CreateThruster](#), [CreateThrusterGroup](#), [AddExhaust](#)

**16.51.3.146 void VESSEL::ClearThrusterDefinitions () const**

Delete all thruster and thruster group definitions.

**Note:**

This function removes all thruster definitions, as well as all the thruster group definitions.  
It also removes all previously defined exhaust render definitions.

**See also:**

[CreateThruster](#), [DelThruster](#), [CreateThrusterGroup](#), [AddExhaust](#)

**16.51.3.147 DWORD VESSEL::GetThrusterCount () const**

Returns the number of thrusters currently defined.

**Returns:**

Number of logical thruster definitions.

**See also:**

[CreateThruster](#), [GetThrusterHandleByIndex](#)

**16.51.3.148 THRUSTER\_HANDLE VESSEL::GetThrusterHandleByIndex (DWORD *idx*) const**

Returns the handle of a thruster specified by its index.

**Parameters:**

*idx* thruster index ( $\geq 0$ )

**Returns:**

Thruster handle

**Note:**

The index must be in the range between 0 and nthruster-1, where nthruster is the thruster count returned by [GetThrusterCount\(\)](#). If the index is out of range, the returned handle is NULL.

The index of a given thruster may change if vessel thrusters are deleted. The handle remains valid until the corresponding thruster is deleted.

**See also:**

[CreateThruster](#), [DelThruster](#), [GetThrusterCount](#)

**16.51.3.149 PROPELLANT\_HANDLE VESSEL::GetThrusterResource (THRUSTER\_HANDLE *th*) const**

Returns a handle for the propellant resource feeding the thruster.

**Parameters:**

*th* thruster handle

**Returns:**

Propellant resource handle, or NULL if the thruster is not connected.

**See also:**

[SetThrusterResource](#), [CreateThruster](#)

**16.51.3.150 void VESSEL::SetThrusterResource (THRUSTER\_HANDLE *th*, PROPELLANT\_HANDLE *ph*) const**

Connect a thruster to a propellant resource.

**Parameters:**

*th* thruster handle

*ph* propellant resource handle

**Note:**

A thruster can only be connected to one propellant resource at a time. Setting a new resource disconnects from the previous resource.

To disconnect the thruster from its current tank, use *ph* = NULL.

**See also:**

[GetThrusterResource](#)

**16.51.3.151 void VESSEL::GetThrusterRef (THRUSTER\_HANDLE *th*, VECTOR3 & *pos*) const**

Returns the thrust force attack point of a thruster.

**Parameters:**

← *th* thruster handle

→ *pos* thrust attack point [m]

**Note:**

*pos* is returned in the vessel frame of reference.

**See also:**

[SetThrusterRef](#), [GetThrusterDir](#)

**16.51.3.152 void VESSEL::SetThrusterRef (THRUSTER\_HANDLE *th*, const VECTOR3 & *pos*) const**

Reset the thrust force attack point of a thruster.

**Parameters:**

*th* thruster handle  
*pos* new force attack point [m]

**Note:**

*pos* is specified in the vessel reference system.

This method should be used whenever a thruster has been physically moved in the vessel's local frame of reference.

If the vessel's centre of gravity, i.e. the origin of its reference system, is moved with [ShiftCG\(\)](#), the thruster positions are updated automatically.

The attack point has no influence on the linear force exerted on the vessel by the thruster, but it affects the induced torque.

**See also:**

[GetThrusterRef](#), [CreateThruster](#), [ShiftCG](#), [SetThrusterDir](#)

**16.51.3.153 void VESSEL::GetThrusterDir (THRUSTER\_HANDLE *th*, VECTOR3 & *dir*) const**

Returns the force direction of a thruster.

**Parameters:**

← *th* thruster handle  
→ *dir* thrust direction (vessel frame of reference)

**See also:**

[SetThrusterDir](#), [GetThrusterRef](#)

**16.51.3.154 void VESSEL::SetThrusterDir (THRUSTER\_HANDLE *th*, const VECTOR3 & *dir*) const**

Reset the force direction of a thruster.

**Parameters:**

*th* thruster handle  
*dir* new thrust direction (vessel frame of reference)

**Note:**

This method can be used to realise a tilt of the rocket motor (e.g. for implementing a thruster gimbal mechanism)

**See also:**

[GetThrusterDir](#), [CreateThruster](#), [SetThrusterRef](#)

**16.51.3.155 double VESSEL::GetThrusterMax0 (THRUSTER\_HANDLE *th*) const**

Returns the maximum vacuum thrust rating of a thruster.

**Parameters:**

*th* thruster handle

**Returns:**

Maximum vacuum thrust rating [N]

**Note:**

To retrieve the actual current maximum thrust rating (which may be lower in the presence of ambient atmospheric pressure), use [GetThrusterMax\(\)](#).

**See also:**

[SetThrusterMax0](#),  
[GetThrusterMax\(THRUSTER\\_HANDLE\)const](#),  
[GetThrusterMax\(THRUSTER\\_HANDLE,double\)const](#)

**16.51.3.156 void VESSEL::SetThrusterMax0 (THRUSTER\_HANDLE *th*, double *maxth0*) const**

Reset the maximum vacuum thrust rating of a thruster.

**Parameters:**

*th* thruster handle  
*maxth0* new maximum vacuum thrust rating [N]

**Note:**

The max. thrust rating in the presence of atmospheric ambient pressure may be lower than the vacuum thrust if a pressure-dependent Isp value has been defined.

**See also:**

[GetThrusterMax0](#), [CreateThruster](#),  
[SetThrusterIsp\(THRUSTER\\_HANDLE,double\)const](#),  
[SetThrusterIsp\(THRUSTER\\_HANDLE,double,double,double\)const](#)

**16.51.3.157 double VESSEL::GetThrusterMax (THRUSTER\_HANDLE *th*) const**

Returns the current maximum thrust rating of a thruster.

**Parameters:**

*th* thruster handle

**Returns:**

Max. thrust rating at the current atmospheric pressure [N].

**Note:**

If a pressure-dependent Isp rating has been defined for the thruster, and if the vessel is moving through a planetary atmosphere, this method returns the maximum thrust rating given the current atmospheric pressure.

Otherwise it returns the maximum vacuum thrust rating of the thruster.

**See also:**

[GetThrusterMax\(THRUSTER\\_HANDLE,double\)const](#),  
[SetThrusterIsp\(THRUSTER\\_HANDLE,double\)const](#),  
[SetThrusterIsp\(THRUSTER\\_HANDLE,double,double,double\)const](#),  
[CreateThruster](#)

**16.51.3.158 double VESSEL::GetThrusterMax (THRUSTER\_HANDLE *th*, double *p\_ref*) const**

Returns the maximum thrust rating of a thruster at a specific ambient pressure.

**Parameters:**

*th* thruster handle  
*p\_ref* reference pressure [Pa]

**Returns:**

Max. thrust rating at atmospheric pressure *p\_ref* [N].

**Note:**

If a pressure-dependent Isp rating has been defined for the thruster, and if the vessel is moving through a planetary atmosphere, this method returns the maximum thrust rating at ambient pressure *p\_ref*. Otherwise it returns the maximum vacuum thrust rating of the thruster.

**See also:**

[GetThrusterMax\(THRUSTER\\_HANDLE\)const](#),  
[SetThrusterIsp\(THRUSTER\\_HANDLE,double\)const](#),  
[SetThrusterIsp\(THRUSTER\\_HANDLE,double,double,double\)const](#),  
[CreateThruster](#)

**16.51.3.159 double VESSEL::GetThrusterIsp0 (THRUSTER\_HANDLE *th*) const**

Returns the vacuum fuel-specific impulse (Isp) rating for a thruster.

**Parameters:**

*th* thruster handle

**Returns:**

Isp value in vacuum [m/s]

**Note:**

Equivalent to GetThrusterIsp (*th*,0)

**See also:**

[GetThrusterIsp\(THRUSTER\\_HANDLE\)const](#),  
[GetThrusterIsp\(THRUSTER\\_HANDLE,double\)const](#),  
[SetThrusterIsp\(THRUSTER\\_HANDLE,double\)const](#),  
[SetThrusterIsp\(THRUSTER\\_HANDLE,double,double,double\)const](#),  
[CreateThruster](#)

**16.51.3.160 double VESSEL::GetThrusterIsp (THRUSTER\_HANDLE *th*) const**

Returns the current fuel-specific impulse (Isp) rating of a thruster.

**Parameters:**

*th* thruster handle

**Returns:**

Current Isp value [m/s].

**Note:**

If the vessel is moving within a planetary atmosphere, and if a pressure-dependent Isp rating has been defined for this thruster, the returned Isp value will vary with ambient atmospheric pressure.

**See also:**

[GetThrusterIsp\(THRUSTER\\_HANDLE,double\)const](#),  
[GetThrusterIsp0](#),  
[SetThrusterIsp\(THRUSTER\\_HANDLE,double\)const](#),  
[SetThrusterIsp\(THRUSTER\\_HANDLE,double,double,double\)const](#),  
[CreateThruster](#)

**16.51.3.161 double VESSEL::GetThrusterIsp (THRUSTER\_HANDLE *th*, double *p\_ref*) const**

Returns the fuel-specific impulse (Isp) rating of a thruster at a specific ambient atmospheric pressure.

**Parameters:**

*th* thruster handle

*p\_ref* reference pressure [Pa]

**Returns:**

Isp value at ambient pressure *p\_ref* [m/s].

**Note:**

If no pressure-dependent Isp rating has been defined for this thruster, it will always return the vacuum rating, independent of the specified pressure.

To obtain vacuum Isp rating, set *p\_ref* to 0.

To obtain the Isp rating at (Earth) sea level, set *p\_ref* = 101.4e3.

**See also:**

[GetThrusterIsp\(THRUSTER\\_HANDLE\)const](#),

---

[GetThrusterIsp0](#),  
[SetThrusterIsp\(THRUSTER\\_HANDLE,double\)const](#),  
[SetThrusterIsp\(THRUSTER\\_HANDLE,double,double,double\)const](#),  
[CreateThruster](#)

### 16.51.3.162 void VESSEL::SetThrusterIsp (THRUSTER\_HANDLE *th*, double *isp*) const

Reset the fuel-specific impulse (Isp) rating of a thruster, assuming no pressure dependence.

**Parameters:**

*th* thruster handle  
*isp* new Isp rating [m/s]

**Note:**

The Isp value correlates the propellant mass flow rate  $dm/dt$  with the resulting thrust force  $F$ :  $F = Isp \cdot (dm/dt)$ .

In the engineering literature, fuel-specific impulse is sometimes given in units of time, by dividing the Isp as defined above by the gravitational acceleration  $1g = 9.81 \text{ m/s}^2$ .

The specified Isp value is assumed to be independent of ambient atmospheric pressure. To define a pressure-dependent Isp value, use [SetThrusterIsp\(THRUSTER\\_HANDLE,double,double,double\)const](#).

**See also:**

[SetThrusterIsp\(THRUSTER\\_HANDLE,double,double,double\)const](#),  
[GetThrusterIsp\(THRUSTER\\_HANDLE\)const](#),  
[GetThrusterIsp\(THRUSTER\\_HANDLE,double\)const](#), [GetThrusterIsp0](#),  
[CreateThruster](#)

### 16.51.3.163 void VESSEL::SetThrusterIsp (THRUSTER\_HANDLE *th*, double *isp0*, double *isp\_ref*, double *p\_ref* = 101.4e3) const

Reset the fuel-specific impulse (Isp) rating of a thruster including a pressure dependency.

**Parameters:**

*th* thruster handle  
*isp0* vacuum Isp rating [m/s]  
*isp\_ref* Isp rating at ambient pressure *p\_ref* [m/s]  
*p\_ref* reference pressure [Pa] for *isp\_ref* (defaults to Earth sea level pressure)

**Note:**

See [SetThrusterIsp\(THRUSTER\\_HANDLE,double\)const](#) for a definition of the relationship between Isp, thrust and fuel mass flow rate.

**See also:**

[SetThrusterIsp\(THRUSTER\\_HANDLE,double\)const](#),  
[GetThrusterIsp\(THRUSTER\\_HANDLE\)const](#),  
[GetThrusterIsp\(THRUSTER\\_HANDLE,double\)const](#),  
[GetThrusterIsp0](#), [CreateThruster](#)

**16.51.3.164 double VESSEL::GetThrusterLevel (THRUSTER\_HANDLE *th*) const**

Returns the current thrust level setting of a thruster.

**Parameters:**

*th* thruster handle

**Returns:**

Current thrust level (0...1)

**Note:**

To obtain the actual force [N] currently generated by the thruster, multiply the thrust level with the max. thrust rating returned by [GetThrusterMax\(\)](#).

**See also:**

[GetThrusterMax](#), [SetThrusterLevel](#)

**16.51.3.165 void VESSEL::SetThrusterLevel (THRUSTER\_HANDLE *th*, double *level*) const**

Set thrust level for a thruster.

**Parameters:**

*th* thruster handle

*level* thrust level (0...1)

**Note:**

At level 1 the thruster generates maximum force, as defined by its maxth parameter.

Certain thrusters are controlled directly by Orbiter via primary input controls (e.g. joystick throttle control for main thrusters), which may override this function.

**See also:**

[IncThrusterLevel](#), [GetThrusterLevel](#)

**16.51.3.166 void VESSEL::IncThrusterLevel (THRUSTER\_HANDLE *th*, double *dlevel*) const**

Apply a change to the thrust level of a thruster.

**Parameters:**

*th* thruster handle

*dlevel* thrust level change (-1...1)

**Note:**

The applied thrust level change is limited to give a resulting thrust level in the range (0...1).

**See also:**

[SetThrusterLevel](#), [GetThrusterLevel](#)

**16.51.3.167 void VESSEL::SetThrusterLevel\_SingleStep (THRUSTER\_HANDLE *th*, double *level*) const**

Set the thrust level of a thruster for the current time step only.

**Parameters:**

*th* thruster handle  
*level* thrust level (0...1)

**Note:**

At level 1 the thruster generates maximum force, as defined by its maxth parameter.

This method overrides the thruster's permanent thrust level for the current time step only, so it should normally only be used in the body of the [VESSEL2::clbkPreStep\(\)](#) method.

**See also:**

[SetThrusterLevel](#), [VESSEL2::clbkPreStep\(\)](#)

**16.51.3.168 void VESSEL::IncThrusterLevel\_SingleStep (THRUSTER\_HANDLE *th*, double *dlevel*) const**

Apply a thrust level change to a thruster for the current time step only.

**Parameters:**

*th* thruster handle  
*dlevel* thrust level change (-1...1)

**Note:**

This method overrides the thruster's permanent thrust level for the current time step only, so it should normally only be used in the body of the [VESSEL2::clbkPreStep\(\)](#) method.

This method may be overridden by manual user input via keyboard and joystick, or by automatic attitude sequences.

The applied thrust level change is limited to give a resulting thrust level in the range (0...1).

**See also:**

[SetThrusterLevel\\_SingleStep](#), [IncThrusterLevel](#), [VESSEL2::clbkPreStep\(\)](#)

**16.51.3.169 void VESSEL::GetThrusterMoment (THRUSTER\_HANDLE *th*, VECTOR3 & *F*, VECTOR3 & *T*) const**

Returns the linear moment (force) and angular moment (torque) currently generated by a thruster.

**Parameters:**

*th* thruster handle  
*F* linear force [N]  
*T* torque [Nm]

**Note:**

The returned values include the influence of ambient pressure on the thrust generated by the engine.

**16.51.3.170 double VESSEL::GetISP () const**

Returns the vessel's current default fuel-specific impulse.

**Returns:**

Fuel-specific impulse [m/s]. This is the amount of thrust [N] obtained by burning 1kg of fuel per second.

**Note:**

The function returns the current default Isp value which will be used for all subsequently defined thrusters which do not define their individual Isp settings.

To obtain an actual Isp value for a thruster, use GetThrusterISP.

The default Isp value can be set by the [SetISP\(\)](#) method, or via the 'Isp' entry in the vessel configuration file. If not defined, the default value is 5e4.

**See also:**

[SetISP](#), [GetThrusterISP](#)

**16.51.3.171 void VESSEL::SetISP (double *isp*) const**

Sets the default Isp value for subsequently created thrusters.

**Parameters:**

*isp* fuel-specific impulse [m/s]

**Note:**

The Isp value defines the amount of thrust [N] obtained by burning 1 kg of fuel per second.

Resetting the default Isp value affects only thrusters which are created subsequently, and which don't define individual Isp values.

Before the first call to SetISP, the initial value is read from the 'Isp' entry of the vessel definition file. If no entry exists, a value of 5e4 is used.

It is recommended to define individual Isp values during thruster creation instead of using SetISP.

**16.51.3.172 THGROUP\_HANDLE VESSEL::CreateThrusterGroup (THRUSTER\_HANDLE \* *th*, int *nth*, THGROUP\_TYPE *thgt*) const**

Combine thrusters into a logical group.

**Parameters:**

*th* array of thruster handles to form a group

*nth* number of thrusters in the array

*thgt* thruster group type (see [Thruster and thruster-group parameters](#))

**Returns:**

thruster group handle

**Note:**

Thruster groups (except for user-defined groups) are engaged by Orbiter as a result of user input. For example, pushing the stick backward in rotational attitude mode will engage the thrusters in the THGROUP\_ATT\_PITCHUP group.

It is the responsibility of the vessel designer to make sure that the thruster groups are designed so that they behave in a sensible way.

Thrusters can be added to more than one group. For example, an attitude thruster can be simultaneously grouped into THGROUP\_ATT\_PITCHUP and THGROUP\_ATT\_UP.

Rotational thrusters should be designed so that they don't induce a significant linear momentum. This means rotational groups require at least 2 thrusters each.

Linear thrusters should be designed such that they don't induce a significant angular momentum.

If a vessel does not define a complete set of attitude thruster groups, certain navmode sequences (e.g. KILLROT) may fail.

**See also:**

[DelThrusterGroup](#), [CreateThruster](#), Thruster and thruster-group parameters

**16.51.3.173 bool VESSEL::DelThrusterGroup (THGROUP\_HANDLE *thg*, bool *delth* = false) const**

Delete a thruster group and (optionally) all associated thrusters.

**Parameters:**

*thg* thruster group handle

*delth* thruster destruction flag (see notes)

**Returns:**

*true* on success.

**Note:**

If *delth==true*, all thrusters associated with the group will be destroyed. Note that this can have side effects if the thrusters were associated with multiple groups, since they are removed from all those groups as well.

**See also:**

[DelThrusterGroup\(THGROUP\\_TYPE,bool\)const](#), [CreateThrusterGroup](#), [DelThruster](#), Thruster and thruster-group parameters

**16.51.3.174 bool VESSEL::DelThrusterGroup (THGROUP\_TYPE *thgt*, bool *delth* = false) const**

Delete a default thruster group and (optionally) all associated thrusters.

**Parameters:**

*thgt* thruster group type (excluding THGROUP\_USER)

*delth* thruster destruction flag

**Returns:**

*true* on success

**Note:**

This version can only be used for default thruster groups (< THGROUP\_USER).

If *delth==true*, all thrusters associated with the group will be destroyed. Note that this can have side effects if the thrusters were associated with multiple groups, since they are removed from all those groups as well.

**See also:**

[DelThrusterGroup\(THGROUP\\_HANDLE,bool\)const](#), [CreateThrusterGroup](#), [DelThruster](#), [Thruster](#) and [thruster-group parameters](#)

**16.51.3.175 THGROUP\_HANDLE VESSEL::GetThrusterGroupHandle (THGROUP\_TYPE *thgt*) const**

Returns the handle of a default thruster group.

**Parameters:**

*thgt* thruster group type (see [Thruster and thruster-group parameters](#))

**Returns:**

thruster group handle (or NULL if no group is defined for the specified type).

**Note:**

The thruster group type must not be THGROUP\_USER. To retrieve the handle of a nonstandard thruster group, use  [GetUserThrusterGroupHandleByIndex\(\)](#).

**See also:**

[GetUserThrusterGroupHandleByIndex](#)

**16.51.3.176 THGROUP\_HANDLE VESSEL:: GetUserThrusterGroupHandleByIndex (DWORD *idx*) const**

Returns the handle of a user-defined (nonstandard) thruster group.

**Parameters:**

*idx* index of user-defined thruster group ( $\geq 0$ )

**Returns:**

thruster group handle (or NULL if index out of range)

**Note:**

Use this method only to retrieve handles for nonstandard thruster groups (created with the THGROUP\_USER flag). For standard groups, use [GetThrusterGroupHandle\(\)](#) instead.

The index must be in the range between 0 and nuserthgroup-1, where nuserthgroup is the number of nonstandard thruster groups. Use  [GetUserThrusterGroupCount\(\)](#) to obtain this value.

**See also:**

[GetThrusterGroupHandle](#),  [GetUserThrusterGroupCount](#)

**16.51.3.177 DWORD VESSEL::GetGroupThrusterCount (THGROUP\_HANDLE *thg*) const**

Returns the number of thrusters assigned to a logical thruster group.

**Parameters:**

*thg* thruster group handle

**Returns:**

Number of thrusters assigned to the specified thruster group.

**Note:**

Thrusters can be assigned to more than one group (and some thrusters may not be assigned to any group) so the sum of GetGroupThrusterCount values over all groups can be different to the total number of thrusters.

**See also:**

[GetGroupThrusterCount\(THGROUP\\_TYPE\)const](#)

**16.51.3.178 DWORD VESSEL::GetGroupThrusterCount (THGROUP\_TYPE *thgt*) const**

Returns the number of thrusters assigned to a standard logical thruster group.

**Parameters:**

*thgt* thruster group enumeration type (see [Thruster and thruster-group parameters](#))

**Returns:**

Number of thrusters assigned to the specified thruster group.

**Note:**

This function only works for standard group types. Do not use it with THGROUP\_USER. For user-defined groups, use [VESSEL::GetGroupThrusterCount\(THGROUP\\_HANDLE\)const](#) instead.

Thrusters can be assigned to more than one group (and some thrusters may not be assigned to any group) so the sum of GetGroupThrusterCount values over all groups can be different to the overall number of thrusters.

**See also:**

[GetGroupThrusterCount\(THGROUP\\_HANDLE\)const](#)

**16.51.3.179 THRUSTER\_HANDLE VESSEL::GetGroupThruster (THGROUP\_HANDLE *thg*, DWORD *idx*) const**

Returns a handle for a thruster that belongs to a specified thruster group.

**Parameters:**

*thg* thruster group handle  
*idx* thruster index ( $0 \leq idx < \text{GetGroupThrusterCount}()$ )

**Returns:**

Thuster handle

**16.51.3.180 THRUSTER\_HANDLE VESSEL::GetGroupThruster (THGROUP\_TYPE *thgt*, DWORD *idx*) const**

Returns a handle for a thruster that belongs to a standard thruster group.

**Parameters:**

*thgt* thruster group enumeration type (see [Thruster and thruster-group parameters](#))  
*idx* thruster index ( $0 \leq idx < \text{GetGroupThrusterCount}()$ )

**Returns:**

Thruster handle

**Note:**

This function only works for standard group types. Do not use with THGROUP\_USER. For user-defined groups, use [GetGroupThruster\(THRUSTER\\_HANDLE,DWORD\)const](#).

**16.51.3.181 DWORD VESSEL:: GetUserThrusterGroupCount () const**

Returns the number of user-defined (nonstandard) thruster groups.

**Returns:**

Number of user-defined thruster groups.

**Note:**

The value returned by this method only includes user-defined thruster groups (created with the THGROUP\_USER flag). It does not contain any standard thruster groups (such as THGROUP\_MAIN, etc.)

**16.51.3.182 bool VESSEL::ThrusterGroupDefined (THGROUP\_TYPE *thgt*) const**

Indicates if a default thruster group is defined by the vessel.

**Parameters:**

*thgt* thruster group enumeration type (see [Thruster and thruster-group parameters](#))

**Returns:**

*true* if the group contains any thrusters, *false* otherwise.

**Note:**

This method only works for default groups. Do not use with THGROUP\_USER.  
A group is considered to be "defined" if it contains at least one thruster.

**See also:**

[GetGroupThrusterCount](#)

**16.51.3.183 void VESSEL::SetThrusterGroupLevel (THGROUP\_HANDLE *thg*, double *level*) const**

Sets the thrust level for all thrusters in a group.

**Parameters:**

*thg* thruster group identifier  
*level* new thrust level (range 0-1)

**See also:**

[SetThrusterGroupLevel\(THGROUP\\_TYPE,double\)const](#)

**16.51.3.184 void VESSEL::SetThrusterGroupLevel (THGROUP\_TYPE *thgt*, double *level*) const**

Sets the thrust level for all thrusters in a standard group.

**Parameters:**

*thgt* thruster group type (see [Thruster and thruster-group parameters](#))  
*level* new thrust level (range 0-1)

**Note:**

This method can only be used with standard thruster group types. Do not use with THGROUP\_USER.

**See also:**

[SetThrusterGroupLevel \(THGROUP\\_HANDLE,double\)const](#)

**16.51.3.185 void VESSEL::IncThrusterGroupLevel (THGROUP\_HANDLE *thg*, double *dlevel*) const**

Increments the thrust level for all thrusters in a group.

**Parameters:**

*thg* thruster group identifier  
*dlevel* thrust level increment

**Note:**

Resulting thrust levels are automatically truncated to the range [0..1]  
Use negative *dlevel* to decrement the thrust level.

**See also:**

[VESSEL::IncThrusterGroupLevel\(THGROUP\\_TYPE,double\)const](#)

**16.51.3.186 void VESSEL::IncThrusterGroupLevel (THGROUP\_TYPE *thgt*, double *dlevel*) const**

Increments the thrust level for all thrusters in a standard group.

**Parameters:**

*thgt* thruster group type

*dlevel* thrust level increment

**Note:**

This method can be used for standard thruster group types enumerated in [Thruster and thruster-group parameters](#) except THGROUP\_USER.

Resulting thrust levels are automatically truncated to the range [0..1]

Use negative *dlevel* to decrement the thrust level.

**See also:**

[VESSEL::IncThrusterGroupLevel\(THGROUP\\_HANDLE,double\)const](#)

**16.51.3.187 void VESSEL::IncThrusterGroupLevel\_SingleStep (THGROUP\_HANDLE *thg*, double *dlevel*) const**

Increments the thrust level of a group for a single time step.

**Parameters:**

*thg* thruster group identifier

*dlevel* thrust level increment

**Note:**

The total thrust level of a thruster group is composed of the sum of a *permanent* and an *override* portion, constrained to range [0..1]. The permanent setting only changes when reset explicitly, while the override setting is reset to zero after each time step.

This function increments the override portion of the thrust level for the thruster group for the current time step only.

Negative values for the override thrust level are permitted to reduce the total thrust level below its permanent setting (down to a minimum of 0).

Any override adjustments of individual thrusters in the group with [IncThrusterLevel\\_SingleStep](#) are added to their total level.

**See also:**

[IncThrusterGroupLevel\\_SingleStep\(THGROUP\\_TYPE,double\)const](#), [IncThrusterLevel\\_SingleStep](#)

**16.51.3.188 void VESSEL::IncThrusterGroupLevel\_SingleStep (THGROUP\_TYPE *thgt*, double *dlevel*) const**

Increments the thrust level of a standard group for a single time step.

**Parameters:**

*thgt* thruster group type

*dlevel* thrust level increment

**Note:**

The total thrust level of a thruster group is composed of the sum of a *permanent* and an *override* portion, constrained to range [0..1]. The permanent setting only changes when reset explicitly, while the override setting is reset to zero after each time step.

This function increments the override portion of the thrust level for the thruster group for the current time step only.

Negative values for the override thrust level are permitted to reduce the total thrust level below its permanent setting (down to a minimum of 0).

Any override adjustments of individual thrusters in the group with [IncThrusterLevel\\_SingleStep](#) are added to their total level.

**See also:**

[IncThrusterGroupLevel\\_SingleStep\(THGROUP\\_HANDLE,double\)const](#), [IncThrusterLevel\\_SingleStep](#)      [IncThrusterLevel\\_-](#)

**16.51.3.189 double VESSEL::GetThrusterGroupLevel (THGROUP\_HANDLE *thg*) const**

Returns the mean thrust level for a thruster group.

**Parameters:**

*thg* thruster group identifier

**Returns:**

Mean group thrust level [0..1]

**Note:**

In general, this method is only useful for groups where all thrusters have the same maximum thrust rating and the same thrust direction.

**See also:**

[GetThrusterGroupLevel\(THGROUP\\_TYPE\)const](#)

**16.51.3.190 double VESSEL::GetThrusterGroupLevel (THGROUP\_TYPE *thgt*) const**

Returns the mean thrust level for a default thruster group.

**Parameters:**

*thgt* thruster group type

**Returns:**

Mean group thrust level [0..1]

**Note:**

In general, this method is only useful for groups where all thrusters have the same maximum thrust rating and the same thrust direction.

**See also:**

[GetThrusterGroupLevel\(THGROUP\\_HANDLE\)const](#)

**16.51.3.191 double VESSEL::GetManualControlLevel (THGROUP\_TYPE *thgt*, DWORD *mode* = MANCTRL\_ATTMODE, DWORD *device* = MANCTRL\_ANYDEVICE) const**

Returns the thrust level of an attitude thruster group set via keyboard or mouse input.

**Parameters:**

*thgt* thruster group identifier

*mode* attitude control mode (see [Manual control mode identifiers](#))

*device* input device (see [Manual control device identifiers](#))

**Returns:**

Manual thrust level [0..1]

**Note:**

If *mode* is not MANCTRL\_ANYMODE, only thruster groups which are of the specified mode (linear or rotational) will return nonzero values.

**16.51.3.192 int VESSEL::GetAttitudeMode () const**

Returns the current RCS (reaction control system) thruster mode.

**Returns:**

Current RCS mode (see [RCS mode identifiers](#))

**Note:**

The reaction control system consists of a set of small thrusters arranged around the vessel. They can be fired in pre-defined configurations to provide either a change in angular velocity (in RCS\_ROT mode) or in linear velocity (in RCS\_LIN mode).

RCS\_NONE indicates that the RCS is disabled or not available.

Currently Orbiter doesn't allow simultaneous linear and rotational RCS control via keyboard or joystick. The user has to switch between the two. However, simultaneous operation is possible via the "RControl" plugin module.

Not all vessel classes may define a complete RCS.

**See also:**

[SetAttitudeMode](#), [RCS mode identifiers](#)

**16.51.3.193 bool VESSEL::SetAttitudeMode (int *mode*) const**

Sets the vessel's RCS (reaction control system) thruster mode.

**Parameters:**

*mode* New RCS mode (see [RCS mode identifiers](#))

**Returns:**

true on success, false for invalid argument

**Note:**

The reaction control system consists of a set of small thrusters arranged around the vessel. They can be fired in pre-defined configurations to provide either a change in angular velocity (in RCS\_ROT mode) or in linear velocity (in RCS\_LIN mode).

Set RCS\_NONE to disable the RCS.

**See also:**

[GetAttitudeMode](#), [RCS mode identifiers](#)

**16.51.3.194 int VESSEL::ToggleAttitudeMode () const**

Switch between linear and rotational RCS mode.

**Returns:**

New RCS mode index

**Note:**

If the RCS is disabled, this method does nothing and returns 0.

During playback, this method does nothing and returns the current RCS mode.

**See also:**

[SetAttitudeMode](#), [GetAttitudeMode](#)

**16.51.3.195 void VESSEL::GetAttitudeRotLevel (VECTOR3 & th) const**

Returns the current combined thrust levels for the reaction control system thruster groups in rotational mode.

**Parameters:**

→ *th* vector containing RCS thruster group levels for rotation around the 3 principal vessel axes (values: -1 to +1).

**Note:**

The fractional thrust levels of the RCS engines for rotation around the vessel's x, y and z axis are returned in the x, y, and z components of *th*, respectively.

The orientation of the vessel axes is implementation-dependent, but usually by convention, +x is "right", +y is "up", and +z is "forward".

A value of +1 denotes maximum thrust in the positive direction around an axis, while -1 denotes maximum thrust in the negative direction.

This method combines the results of calls to GetThrusterGroupLevel for all relevant RCS thruster groups in the following combinations:

th.x	THGROUP_ATT_PITCHUP - THGROUP_ATT_PITCHDOWN
th.y	THGROUP_ATT_YAWLEFT - THGROUP_ATT_YAWRIGHT
th.z	THGROUP_ATT_BANKRIGHT - THGROUP_ATT_BANKLEFT

To obtain the actual thrust force magnitudes [N], the absolute values must be multiplied with the max. attitude thrust.

**See also:**

[GetAttitudeLinLevel](#), [SetAttitudeRotLevel](#), [GetThrusterGroupLevel](#), [GetAttitudeMode](#)

**16.51.3.196 void VESSEL::SetAttitudeRotLevel (const VECTOR3 & *th*) const**

Set RCS thruster levels for rotation in all 3 vessel axes.

**Parameters:**

*th* RCS thruster levels for rotation around x,y,z axes (range -1...+1)

**Note:**

This method is functional even if the manual RCS input mode is set to linear.

**See also:**

[SetAttitudeRotLevel\(int,double\)const](#), [GetAttitudeRotLevel](#), [SetAttitudeLinLevel](#)

**16.51.3.197 void VESSEL::SetAttitudeRotLevel (int *axis*, double *th*) const**

Set RCS thruster level for rotation around a single axis.

**Parameters:**

*axis* rotation axis (0=x, 1=y, 2=z)

*th* RCS thruster level (range -1...+1)

**Note:**

This method is functional even if the manual RCS input mode is set to linear.

**See also:**

[SetAttitudeRotLevel\(const VECTOR3&\)const](#), [GetAttitudeRotLevel](#), [SetAttitudeLinLevel](#)

**16.51.3.198 void VESSEL::GetAttitudeLinLevel (VECTOR3 & *th*) const**

Returns the current combined thrust levels for the reaction control system thruster groups in linear (translational) mode.

**Parameters:**

→ *th* vector containing RCS thruster group levels for translation along the 3 principal vessel axes  
(values: -1 to +1)

**Note:**

The fractional thrust levels of the RCS engines for translation along the vessel's x, y and z axis are returned in the x, y, and z components of *th*, respectively.

The orientation of the vessel axes is implementation-dependent, but usually by convention, +x is "right", +y is "up", and +z is "forward".

A value of +1 denotes maximum thrust in the positive direction along an axis, while -1 denotes maximum thrust in the negative direction.

This method combines the results of calls to GetThrusterGroupLevel for all relevant RCS thruster groups in the following combinations:

th.x	THGROUP_ATT_RIGHT - THGROUP_ATT_LEFT
th.y	THGROUP_ATT_UP - THGROUP_ATT_DOWN
th.z	THGROUP_ATT_FORWARD - THGROUP_ATT_BACK

To obtain the actual thrust force magnitudes [N], the absolute values must be multiplied with the max. attitude thrust.

**See also:**

[GetAttitudeRotLevel](#), [SetAttitudeLinLevel](#), [GetThrusterGroupLevel](#), [GetAttitudeMode](#)

#### 16.51.3.199 void VESSEL::SetAttitudeLinLevel (const VECTOR3 & *th*) const

Set RCS thruster levels for linear translation in all 3 vessel axes.

**Parameters:**

*th* RCS thruster levels (range -1...+1)

**Note:**

This method is functional even if the manual RCS input mode is set to rotational.

**See also:**

[SetAttitudeLinLevel\(int,double\)const](#), [SetAttitudeLinLevel](#), [SetAttitudeRotLevel](#)

#### 16.51.3.200 void VESSEL::SetAttitudeLinLevel (int *axis*, double *th*) const

Set RCS thruster level for linear translation along a single axis.

**Parameters:**

*axis* translation axis (0=x, 1=y, 2=z)

*th* RCS thruster level (range -1...+1)

**Note:**

This method is functional even if the manual RCS input mode is set to rotational.

**See also:**

[SetAttitudeLinLevel\(const VECTOR3&\)const](#), [SetAttitudeLinLevel](#), [SetAttitudeRotLevel](#)

#### 16.51.3.201 int VESSEL::SendBufferedKey (DWORD *key*, bool *down* = true, char \* *kstate* = 0)

Send a simulated buffered key event to the vessel.

**Parameters:**

*key* key code  
*down* key down event flag  
*kstate* key state map for additional modifier keys

**Returns:**

Process flag (0=key not processed, 1=key processed)

**Note:**

This method simulates a manual keyboard press and can be used to trigger actions associated with the key.  
If *down* = true, a key down event is simulated. Otherwise, a key up event is simulated.  
Additional modifier keys (e.g. Ctrl, Shift, Alt) can be set by passing a *kstate* array with the appropriate keys defined.  
This method triggers a call to [VESSEL2::clbkConsumeBufferedKey](#). If not consumed by the callback function, the key event is offered to the default key handler.

**See also:**

[VESSEL2::clbkConsumeBufferedKey](#)

**16.51.3.202 void VESSEL::InitNavRadios (DWORD *nnav*) const**

Defines the number of navigation (NAV) radio receivers supported by the vessel.

**Parameters:**

*nnav* number of NAV radio receivers

**Note:**

A vessel requires NAV radio receivers to obtain instrument navigation aids such as ILS or docking approach information.  
If no NAV receivers are available, then certain [MFD](#) modes such as Landing or Docking will not be supported.  
Default is 2 NAV receivers.

**See also:**

[GetNavCount](#)

**16.51.3.203 DWORD VESSEL::GetNavCount () const**

Returns the number of NAV radio receivers.

**Returns:**

Number of NAV receivers ( $\geq 0$ )

**See also:**

[InitNavRadios](#)

**16.51.3.204 bool VESSEL::SetNavChannel (DWORD *n*, DWORD *ch*) const**

Sets the channel of a NAV radio receiver.

**Parameters:**

*n* receiver index ( $\geq 0$ )

*ch* channel ( $\geq 0$ )

**Returns:**

*false* on error (receiver index or channel out of range), *true* otherwise

**Note:**

NAV radios can be tuned from 108.00 to 139.95 MHz in steps of 0.05 MHz, corresponding to channels 0 to 639.

**See also:**

[InitNavRadios](#), [GetNavChannel](#)

**16.51.3.205 DWORD VESSEL::GetNavChannel (DWORD *n*) const**

Returns the current channel setting of a NAV radio receiver.

**Parameters:**

*n* receiver index ( $\geq 0$ )

**Returns:**

Receiver channel [0..639]. If index *n* is out of range, the return value is 0.

**See also:**

[GetNavRecvFreq](#), [SetNavChannel](#)

**16.51.3.206 float VESSEL::GetNavRecvFreq (DWORD *n*) const**

Returns the current radio frequency of a NAV radio receiver.

**Parameters:**

*n* receiver index ( $\geq 0$ )

**Returns:**

Receiver frequency [MHz] (range 108.00 to 139.95). If index *n* is out of range, the return value is 0.0.

**See also:**

[GetNavChannel](#)

**16.51.3.207 void VESSEL::EnableTransponder (bool *enable*) const**

Enable/disable transmission of transponder signal.

**Parameters:**

*enable* *true* to enable the transponder, *false* to disable.

**Note:**

The transponder is a radio transmitter which can be used by other vessels to obtain navigation information, e.g. for docking rendezvous approaches.

If the transponder is turned on (*enable* = *true*), its initial frequency is set to 108.00 MHz (channel 0). Use [SetTransponderChannel](#) to tune to a different frequency.

**See also:**

[SetTransponderChannel](#), [SetIDSChannel](#)

**16.51.3.208 bool VESSEL::SetTransponderChannel (DWORD *ch*) const**

Switch the channel number of the vessel's transponder.

**Parameters:**

*ch* transponder channel [0..639]

**Returns:**

*false* indicates failure (transponder not enabled or input parameter out of range)

**Note:**

Transponders can be tuned from 108.00 to 139.95 MHz in steps of 0.05 MHz. The frequency corresponding to a channel number *ch* is given by  $f = (108.0 + 0.05 \cdot ch)$  MHz.

**See also:**

[EnableTransponder](#), [SetNavChannel](#)

**16.51.3.209 void VESSEL::EnableIDS (DOCKHANDLE *hDock*, bool *bEnable*) const**

Enable/disable one of the vessel's IDS (Instrument Docking System) transmitters.

**Parameters:**

*hDock* docking port handle

*bEnable* *true* to enable, *false* to disable the IDS for the dock.

**Note:**

If the IDS transmitter is turned on (*bEnable* = *true*), its channel is initially set to 0 (transmitter frequency 108.00 MHz). Use [SetIDSChannel](#) to tune to a different channel.

**See also:**

[SetIDSChannel](#), [EnableTransponder](#)

**16.51.3.210 bool VESSEL::SetIDSChannel (DOCKHANDLE *hDock*, DWORD *ch*) const**

Switch the channel number of one of the vessel's IDS (Instrument Docking System) transmitters.

**Parameters:**

*hDock* docking port handle

*ch* IDS channel [0..639]

**Returns:**

*false* indicates failure (IDS not enabled or input parameter out of range)

**Note:**

IDS transmitters can be tuned from 108.00 to 139.95 MHz in steps of 0.05 MHz. The frequency corresponding to a channel number *ch* is given by  $f = (108.0 + 0.05 \cdot ch)$  MHz.

**See also:**

[EnableIDS](#), [SetTransponderChannel](#), [SetNavChannel](#)

**16.51.3.211 NAVHANDLE VESSEL::GetTransponder () const**

Return handle of vessel transponder if available.

**Returns:**

Navigation radio handle of the vessel's transponder, or NULL if not available.

**Note:**

This function returns NULL unless the transponder has been enabled by a call to [EnableTransponder](#) or by setting the EnableXPDR entry in the vessel's config file to TRUE.

It is not safe to store the handle, because it can become invalid as a result of disabling/enabling the transponder. Instead, the handle should be queried when needed.

The handle can be used to retrieve information about the transmitter, such as current frequency.

**See also:**

[EnableTransponder](#), [SetTransponderChannel](#)

**16.51.3.212 NAVHANDLE VESSEL::GetIDS (DOCKHANDLE *hDock*) const**

Return handle of one of the vessel's instrument docking system (IDS) radio transmitters.

**Parameters:**

*hDock* docking port handle

**Returns:**

Navigation radio handle of the vessel's IDS transmitter for docking port *hDock*.

**Note:**

This function returns NULL if *hDock* does not define an IDS transmitter.

Docking port handles are returned by the [CreateDock](#) and [GetDockHandle](#) methods.

The IDS handle becomes invalid when the dock is deleted (e.g. as a result of [DelDock](#) or [ClearDockDefinitions](#)).

The handle returned by this function can be used to retrieve information about the transmitter, such as sender frequency.

**See also:**

[CreateDock](#), [GetDockHandle](#), [DelDock](#), [ClearDockDefinitions](#), [EnableIDS](#), [GetTransponder](#)

**16.51.3.213 NAVHANDLE VESSEL::GetNavSource (DWORD *n*) const**

Return handle of transmitter source currently received by one of the vessel's NAV receivers.

**Parameters:**

*n* NAV receiver index ( $\geq 0$ )

**Returns:**

handle of transmitter currently received, or NULL if the receiver is not tuned to any station, or if *n* is out of range.

**Note:**

The handle returned by this function may change in consecutive calls, depending on the radio frequency of the corresponding receiver, the vessel position and the position of radio transmitters in the range of the receiver.

**16.51.3.214 void VESSEL::SetCameraOffset (const VECTOR3 & *co*) const**

Set the camera position for internal (cockpit) view.

**Parameters:**

*co* camera offset in vessel coordinates [**m**]

**Note:**

The camera offset can be used to define the pilot's eye position in the spacecraft.

The default offset is (0,0,0).

This function is called typically either globally in [VESSEL2::clbkSetClassCaps](#), if the camera position doesn't change between views, or individually in [VESSEL2::clbkLoadGenericCockpit](#), [VESSEL2::clbkLoadPanel](#) and [VESSEL2::clbkLoadVC](#) for each defined view.

**See also:**

[GetCameraOffset](#)

**16.51.3.215 void VESSEL::GetCameraOffset (VECTOR3 & *co*) const**

Returns the current camera position for internal (cockpit) view.

**Parameters:**

*co* camera offset in vessel coordinates [m]

**See also:**

[SetCameraOffset](#)

**16.51.3.216 void VESSEL::SetCameraDefaultDirection (const VECTOR3 & *cd*) const**

Set the default camera direction for internal (cockpit) view.

**Parameters:**

*cd* new default direction in vessel coordinates

**Note:**

By default, the default direction is (0,0,1), i.e. forward.

The supplied direction vector must be normalised to length 1.

Calling this function automatically sets the current actual view direction to the default direction.

This function can either be called during [VESSEL2::clbkSetClassCaps](#), to define the default camera direction globally for the vessel, or during [VESSEL2::clbkLoadGenericCockpit](#), [VESSEL2::clbkLoadPanel](#) and [VESSEL2::clbkLoadVC](#), to define different default directions for different instrument panels or virtual cockpit positions.

In Orbiter, the user can return to the default direction by pressing the *Home* key on the cursor key pad.

**See also:**

[SetCameraDefaultDirection\(const VECTOR3&,double\)const](#), [GetCameraDefaultDirection](#), [VESSEL2::clbkSetClassCaps](#), [VESSEL2::clbkLoadGenericCockpit](#), [VESSEL2::clbkLoadPanel](#), [VESSEL2::clbkLoadVC](#)

**16.51.3.217 void VESSEL::SetCameraDefaultDirection (const VECTOR3 & *cd*, double *tilt*) const**

Set the default camera direction and tilt angle for internal (cockpit) view.

**Parameters:**

*cd* new default direction in vessel coordinates

*tilt* camera tilt angle around the default direction [rad]

**Note:**

This function allows to set the camera tilt angle in addition to the default direction.

By default, the default direction is (0,0,1), i.e. forward, and the tilt angle is 0 (upright).

The supplied direction vector must be normalised to length 1.

The tilt angle should be in the range [-Pi,+Pi]

Calling this function automatically sets the current actual view direction to the default direction.

**See also:**

[SetCameraDefaultDirection\(const VECTOR3&\)const](#), [GetCameraDefaultDirection](#)

**16.51.3.218 void VESSEL::GetCameraDefaultDirection (VECTOR3 & *cd*) const**

Returns the default camera direction for internal (cockpit) view.

**Parameters:**

*cd* default camera direction in vessel coordinates

**Note:**

The default camera direction may change as a result of invoking SetCameraDefaultDirection, typically when the user selects a different instrument panel or virtual cockpit position.

The returned direction vector is normalised to length 1.

**See also:**

[SetCameraDefaultDirection\(const VECTOR3&\)const](#), [SetCameraDefaultDirection\(const VECTOR3&,double\)const](#)

**16.51.3.219 void VESSEL::SetCameraCatchAngle (double *cangle*) const**

Set the angle over which the cockpit camera auto-centers to default direction.

**Parameters:**

*cangle* auto-center catchment angle [rad]

**Note:**

The cockpit camera auto-centers to its default ("forward") direction when it is close enough to this direction. This function can be used to specify the angle over which auto-centering occurs.

Setting cangle=0 disables the auto-centering function.

The default catchment angle is 5 degrees (5.0\*RAD).

To reset the catchment angle globally for all cockpit views of the vessel, SetCameraCatchAngle would typically be used in [VESSEL2::clbkSetClassCaps\(\)](#). To reset the catchment angle for individual cockpit positions, the function would be used for the appropriate cockpit modes in [VESSEL2::clbkLoadPanel\(\)](#) and [VESSEL2::clbkLoadVC\(\)](#).

**16.51.3.220 void VESSEL::SetCameraRotationRange (double *left*, double *right*, double *up*, double *down*) const**

Sets the range over which the cockpit camera can be rotated from its default direction.

**Parameters:**

*left* rotation range to the left [rad]

*right* rotation range to the right [rad]

*up* rotation range up [rad]

*down* rotation range down [rad]

**Note:**

The meaning of the "left", "right", "up" and "down" directions is given by the orientation of the local vessel frame. For a default view direction of (0,0,1), "left" is a rotation towards the -x axis, "right" is

a rotation towards the +x axis, "up" is a rotation towards the +y axis, and "down" is a rotation towards the -y axis.

All ranges must be  $\geq 0$ . The left and right ranges should be  $< \pi$ . The up and down ranges should be  $< \pi/2$ .

The default values are  $0.8\pi$  for left and right ranges, and  $0.4\pi$  for up and down ranges.

#### See also:

[SetCameraShiftRange](#), [SetCameraMovement](#)

#### 16.51.3.221 void VESSEL::SetCameraShiftRange (const VECTOR3 & *fpos*, const VECTOR3 & *lpos*, const VECTOR3 & *rpos*) const

Set the linear movement range for the cockpit camera.

Defining a linear movement allows the user to move the head forward or sideways, e.g. to get a better look out of a window, or a closer view of a virtual cockpit instrument panel.

#### Parameters:

*fpos* offset vector when leaning forward [m]

*lpos* offset vector when leaning left [m]

*rpos* offset vector when leaning right [m]

#### Note:

If a linear movement range is defined with this function, the user can 'lean' forward or sideways using the 'cockpit slew' keys. Supported keys are:

Name	default	action
CockpitCamDontLean	Ctrl+Alt+Down	return to default position
CockpitCamLeanForward	Ctrl+Alt+Up	lean forward
CockpitCamLeanLeft	Ctrl+Alt+Left	lean left
CockpitCamLeanRight	Ctrl+Alt+Right	lean right

The movement vectors are taken relative to the default cockpit position defined via SetCameraOffset. This function should be called when initialising a cockpit mode (e.g. in clbkLoadPanel or clbkLoadVC). By default, Orbiter resets the linear movement range to zero whenever the cockpit mode changes.

In addition to the linear movement, the camera also turns left when leaning left, turns right when leaning right, and returns to default direction when leaning forward. For more control over camera rotation at the different positions, use SetCameraMovement instead.

#### See also:

[SetCameraMovement](#), [SetCameraRotationRange](#)

#### 16.51.3.222 void VESSEL::SetCameraMovement (const VECTOR3 & *fpos*, double *fphi*, double *fht*, const VECTOR3 & *lpos*, double *lphi*, double *lht*, const VECTOR3 & *rpos*, double *rphi*, double *rht*) const

Set both linear movement range and orientation of the cockpit camera when "leaning" forward, left and right.

#### Parameters:

*fpos* offset vector when leaning forward [m]

*fphi* camera rotation azimuth angle when leaning forward [rad]  
*ftht* camera rotation polar angle when leaning forward [rad]  
*lpos* offset vector when leaning left [**m**]  
*lphi* camera rotation azimuth angle when leaning left [rad]  
*ltht* camera rotation polar angle when leaning left [rad]  
*rpos* offset vector when leaning right [**m**]  
*rphi* camera rotation azimuth angle when leaning right [rad]  
*rtht* camera rotation polar angle when leaning right [rad]

**Note:**

This function is an extended version of [SetCameraShiftRange](#). It is more versatile, because in addition to the linear camera movement vectors, it also allows to define the camera orientation (via azimuth and polar angle relative to default view direction). This allows to point the camera to a particular cockpit window, instrument panel, etc.

**See also:**

[SetCameraShiftRange](#), [SetCameraRotationRange](#)

**16.51.3.223 void VESSEL::ClearMeshes (bool retain\_anim) const**

Remove all mesh definitions for the vessel.

**Parameters:**

*retain\_anim* flag for retaining mesh animation objects

**Note:**

If *retain\_anim* is *false*, all animations defined for the vessel are deleted together with the meshes. If *true*, the animations stay behind. This is only useful if the same meshes are subsequently added again in the same order, so that the animations point to the appropriate meshes and mesh groups and can be re-used. If different meshes are loaded later, the behaviour of the animations becomes undefined. In the future, obsolete method [will](#) be removed, and *retain\_anim* will have a default value of false.

**16.51.3.224 UINT VESSEL::AddMesh (const char \* meshname, const VECTOR3 \* ofs = 0) const**

Load a mesh definition for the vessel from a file.

**Parameters:**

*meshname* mesh file name

*ofs* optional pointer to a displacement vector which describes the offset of the mesh origin against the vessel origin [**m**].

**Returns:**

mesh index

**Note:**

*meshname* defines a path to an existing mesh file. The mesh must be in Orbiter's MSH format (see 3DModel.pdf).

The file name (including optional directory path) is relative to Orbiter's mesh directory (usually ".\\Meshes"). The file extension must not be specified (.msh is assumed.)

The mesh is either appended to the end of the vessel's mesh list, or inserted at the location of a previously deleted mesh (see [VESSEL::DelMesh](#))

The returned value is the mesh list index at which the mesh reference was stored. It can be used to identify the mesh later (e.g. for animations).

This function only creates a *reference* to a mesh, but does not directly load the mesh from file. The mesh is physically loaded only when it is required (whenever the vessel moves within visual range of the observer camera).

**See also:**

[AddMesh\(MESHHANDLE,const VECTOR3\\*\)const](#), [DelMesh](#)

**16.51.3.225 UINT VESSEL::AddMesh (MESHHANDLE *hMesh*, const VECTOR3 \* *ofs* = 0) const**

Add a pre-loaded mesh definition to the vessel.

**Parameters:**

*hMesh* mesh handle

*ofs* optional pointer to a displacement vector which describes the offset of the mesh origin against the vessel origin [**m**].

**Returns:**

mesh index

**Note:**

*hMesh* is a handle to a mesh previously loaded with [oapiLoadMeshGlobal](#).

The global handle *hMesh* represents a "mesh template". Whenever the vessel needs to create its visual representation (when moving within visual range of the observer camera), it creates its individual mesh as a copy of the template.

**See also:**

[AddMesh\(const char\\*,const VECTOR3\\*\)const](#), [oapiLoadMeshGlobal](#), [DelMesh](#)

**16.51.3.226 UINT VESSEL::InsertMesh (const char \* *meshname*, UINT *idx*, const VECTOR3 \* *ofs* = 0) const**

Insert or replace a mesh at a specific index location of the vessel's mesh list.

**Parameters:**

*meshname* mesh file name

*idx* mesh list index ( $\geq 0$ )

*ofs* optional pointer to a displacement vector which describes the offset of the mesh origin against the vessel origin [**m**].

**Returns:**

mesh index

**Note:**

*meshname* defines a path to an existing mesh file. The mesh must be in Orbiter's MSH format. The file name (including optional directory path) is relative to Orbiter's mesh directory (usually ".\\Meshes"). The file extension should not be specified (.msh is assumed.) *idx* is a zero-based index which specifies at which point the mesh reference is added into the vessel's mesh list. If a mesh already exists at this position, it is overwritten. If *idx* > number of meshes, then the required number of (empty) entries is generated. The return value is always equal to *idx*.

**See also:**

[InsertMesh\(MESHHANDLE,UINT,const VECTOR3\\*\)const](#), [AddMesh\(const char\\*,const VECTOR3\\*\)const](#), [AddMesh\(MESHHANDLE,const VECTOR3\\*\)const](#)

**16.51.3.227 UINT VESSEL::InsertMesh (MESHHANDLE *hMesh*, UINT *idx*, const VECTOR3 \**ofs* = 0) const**

Insert or replace a mesh at a specific index location of the vessel's mesh list.

**Parameters:**

*hMesh* mesh handle

*idx* mesh list index ( $\geq 0$ )

*ofs* optional pointer to a displacement vector which describes the offset of the mesh origin against the vessel origin [**m**].

**Returns:**

mesh index

**Note:**

*hMesh* is a handle to a mesh previously loaded with [oapiLoadMeshGlobal](#). The global handle *hMesh* represents a "mesh template". Whenever the vessel needs to create its visual representation (when moving within visual range of the observer camera), it creates its individual mesh as a copy of the template. *idx* is a zero-based index which specifies at which point the mesh reference is added into the vessel's mesh list. If a mesh already exists at this position, it is overwritten. If *idx* > number of meshes, then the required number of (empty) entries is generated. The return value is always equal to *idx*.

**See also:**

[InsertMesh\(const char\\*,UINT,const VECTOR3\\*\)const](#), [AddMesh\(const char\\*,const VECTOR3\\*\)const](#), [AddMesh\(MESHHANDLE,const VECTOR3\\*\)const](#)

**16.51.3.228 bool VESSEL::DelMesh (UINT *idx*, bool *retain\_anim* = false) const**

Remove a mesh from the vessel's mesh list.

**Parameters:**

*idx* mesh list index ( $\geq 0$ )  
*retain\_anim* flag for keeping mesh animations

**Returns:**

*true* on success, *false* to indicate failure (index out of range, or mesh already deleted.)

**Note:**

After a mesh has been deleted, the mesh index is no longer valid, and should not be used any more in function calls (e.g. for animations).

If meshes are added subsequently, they are placed in the vacant list slots, and therefore can be assigned the indices of previously deleted meshes.

If you want to replace a mesh, it is easier to use the [InsertMesh](#) function instead of a combination of [DelMesh](#) and [AddMesh](#).

By default, all animation components associated with the mesh are deleted. This can be prevented by setting *retain\_anim* to true. In general this is only useful if the same mesh is subsequently loaded again into the same mesh index slot. In all other cases, retaining the animations of deleted meshes can lead to undefined behaviour.

**See also:**

[InsertMesh](#), [AddMesh](#), [ClearMeshes](#)

**16.51.3.229 bool VESSEL::ShiftMesh (UINT *idx*, const VECTOR3 & *ofs*) const**

Shift the position of a mesh relative to the vessel's local coordinate system.

**Parameters:**

*idx* mesh list index ( $\geq 0$ )  
*ofs* translation vector [**m**]

**Returns:**

*true* on success, *false* indicates error (index out of range).

**Note:**

This function does not define an animation (i.e. gradual transition), but resets the mesh position instantly.

**See also:**

[ShiftMeshes](#), [GetMeshOffset](#)

**16.51.3.230 void VESSEL::ShiftMeshes (const VECTOR3 & *ofs*) const**

Shift the position of all meshes relative to the vessel's local coordinate system.

**Parameters:**

*ofs* translation vector [**m**]

**Note:**

This function is useful when resetting a vessel's centre of gravity, in combination with [ShiftCentreOfMass](#).

A more convenient way to shift the centre of gravity is a call to [ShiftCG](#).

**See also:**

[ShiftMesh](#), [GetMeshOffset](#), [ShiftCentreOfMass](#), [ShiftCG](#)

**16.51.3.231 bool VESSEL::GetMeshOffset (UINT *idx*, VECTOR3 & *ofs*) const**

Returns the mesh offset in the vessel frame.

**Parameters:**

*idx* mesh index ( $0 \leq \text{idx} < \text{GetMeshCount}()$ )  
→ *ofs* mesh offset [**m**]

**Returns:**

true if idx refers to a valid mesh index

**See also:**

[AddMesh](#), [InsertMesh](#), [ShiftMesh](#), [ShiftMeshes](#)

**16.51.3.232 UINT VESSEL::GetMeshCount () const**

Number of meshes.

Returns the number of meshes currently defined for the vessel

**Returns:**

mesh count ( $\geq 0$ )

**16.51.3.233 MESHHANDLE VESSEL::GetMesh (VISHANDLE *vis*, UINT *idx*) const**

Obtain mesh handle for a vessel mesh.

Returns a handle for a vessel mesh *instance*. Mesh instances only exist while the vessel is within visual range of the camera. This function should therefore only be called between [VESSEL2::clbkVisualCreated](#) and [VESSEL2::clbkVisualDestroyed](#), with the VISHANDLE provided by these functions.

**Parameters:**

*vis* identifies the visual for which the mesh was created  
*idx* mesh index ( $0 \leq \text{idx} < \text{GetMeshCount}()$ )

**Returns:**

mesh handle

**orbiter\_ng:**

The non-graphics version of Orbiter returns always NULL, even if a graphics client is attached. To obtain a client-specific mesh handle, use [GetDevMesh](#) .

**See also:**

[GetMeshTemplate](#), [GetMeshCount](#), [GetDevMesh](#)

**16.51.3.234 DEVMESSHHANDLE VESSEL::GetDevMesh (VISHANDLE *vis*, UINT *idx*) const**

Returns a handle for a device-specific mesh instance.

**Parameters:**

*vis* identifies the visual for which the mesh was created.

*idx* mesh index ( $0 \leq idx < \text{GetMeshCount}()$ )

**Returns:**

device mesh handle

**Note:**

For Orbiter\_ng, this method returns a handle for a mesh instance managed by the external graphics client. Graphics clients may implement their own mesh formats, so the object pointed to by the handle is client-specific.

For inline graphics version, the returned handle points to the same object as the handle returned by [GetMesh](#).

**See also:**

[GetMesh](#)

**16.51.3.235 const MESHHANDLE VESSEL::GetMeshTemplate (UINT *idx*) const**

Obtain a handle for a vessel mesh template.

Returns the mesh handle for a pre-loaded mesh template, if available.

**Parameters:**

*idx* mesh index ( $0 \leq idx < \text{GetMeshCount}()$ )

**Returns:**

mesh template handle

**Note:**

Mesh templates can only be returned for meshes pre-loaded with [oapiLoadMeshGlobal\(\)](#). For all other (load-on-demand) meshes this method returns NULL.

Mesh templates are resources shared between all vessels and should never be modified by individual vessels. Orbiter creates individual copies of the templates whenever a vessel is rendered.

**16.51.3.236 const char\* VESSEL::GetMeshName (UINT *idx*) const**

Obtain mesh file name for an on-demand mesh.

Returns the mesh file name (with path relative to Orbiter's main mesh directory) for a vessel mesh that is loaded on demand (i.e. not pre-loaded).

**Parameters:**

*idx* mesh index ( $0 \leq idx < \text{GetMeshCount}()$ )

**Returns:**

mesh file name, or NULL if mesh is pre-loaded

**Note:**

The file names for pre-loaded meshes are not retained by Orbiter. Graphics clients can obtain pre-loaded mesh file names by intercepting the [oapi::GraphicsClient::clbkStoreMeshPersistent\(\)](#) method.

**16.51.3.237 MESHHANDLE VESSEL::CopyMeshFromTemplate (UINT *idx*) const**

Make a copy of one of the vessel's mesh templates.

**Parameters:**

*idx* mesh index

**Returns:**

handle of copied mesh

**Note:**

Meshes loaded with [oapiLoadMeshGlobal](#) are templates shared between all vessel instances and should never be modified by individual vessels. If a vessel needs to modify its meshes, it should operate on a copy of the template.

**16.51.3.238 WORD VESSEL::GetMeshVisibilityMode (UINT *idx*) const**

Returns the visibility flags for a vessel mesh.

**Parameters:**

*idx* mesh index ( $\geq 0$ )

**Returns:**

Visibility mode flags (see [SetMeshVisibilityMode](#) for possible values).

**See also:**

[SetMeshVisibilityMode](#), Vessel mesh visibility flags

**16.51.3.239 void VESSEL::SetMeshVisibilityMode (UINT *idx*, WORD *mode*) const**

Set the visibility flags for a vessel mesh.

**Parameters:**

*idx* mesh index ( $\geq 0$ )

*mode* visibility mode flags (see [Vessel mesh visibility flags](#))

**Note:**

This method can be used to specify if a mesh is visible in particular camera modes. Some meshes may only be visible in external views, while others should only be visible in cockpit views.

Turning off the unnecessary rendering of meshes can improve the performance of the simulator.

*mode* can be a combination of the [Vessel mesh visibility flags](#).

The default mode after adding a mesh is MESHVIS\_EXTERNAL.

MESHVIS\_EXTPASS can't be used on its own, but as a modifier to any of the other visibility modes. If specified, it forces the mesh to be rendered in Orbiter's external render pass, even if it is labelled as internal (e.g. MESHVIS\_COCKPIT or MESHVIS\_VC). The significance of the external render pass is that it allows the mesh to be obscured by other objects in front of it. However, objects in the external render pass are clipped at a camera distance of 2.5m. Meshes that are rendered during the internal pass always cover all other objects, and have a smaller clipping distance.

Use the MESHVIS\_EXTPASS modifier for parts of the vessel that are visible from the cockpit, but are not close to the camera and may be obscured by other objects. An example is the Shuttle payload bay, which can be covered by payload objects.

**See also:**

[GetMeshVisibilityMode](#), [Vessel mesh visibility flags](#)

### 16.51.3.240 bool VESSEL::MeshgroupTransform (VISHANDLE *vis*, const MESHGROUP\_TRANSFORM & *mt*) const

Affine transformation of a mesh group.

**Parameters:**

*vis* vessel visual handle

*mt* transformation parameter structure

**Returns:**

*true* on success, *false* on failure (group index out of range)

**orbiter\_ng:**

This function is not yet supported in orbiter\_ng and always returns *false*.

### 16.51.3.241 int VESSEL::MeshModified (MESHHANDLE *hMesh*, UINT *grp*, DWORD *modflag*)

Notifies Orbiter of a change in a mesh group.

**Parameters:**

*hMesh* mesh handle

*grp* group index ( $\geq 0$ )

*modflag* type of modification (currently ignored)

**Returns:**

error code (0=ok)

**Note:**

This method should be called if the components of a mesh group (vertices or indices) have been modified, to allow Orbiter to propagate the changes to the render object.

For the built-in renderer, this registration is not strictly necessary, because it uses the mesh directly as the render object, so any changes to the mesh groups are applied directly.

External graphics clients however may map the mesh data into device-specific data structures. In that case, MeshModified tells the graphics subsystem to synchronise its mesh data.

MeshModified does not need to be called after applying an affine transformation of the mesh group as a whole ([MeshgroupTransform](#)), because this is performed by assigning a transformation matrix, rather than by modifying the vertex positions themselves.

**See also:**

[oapiMeshGroup](#), [oapiMeshGroupEx](#)

**16.51.3.242 void VESSEL::RegisterAnimation () const**

Logs a request for calls to [VESSEL2::clbkAnimate](#).

**Note:**

This function allows to implement animation sequences in combination with the VESSEL2clbkAnimate callback function. After a call to RegisterAnimation, VESSEL2clbkAnimate is called at each time step whenever the vessel's visual object exists.

Use [UnregisterAnimation](#) to stop further calls to VESSEL2clbkAnimate.

Each call to RegisterAnimation increments a reference counter, while each call to UnregisterAnimation decrements the counter. Orbiter continues calling VESSEL2clbkAnimate as long as the counter is greater than 0.

If VESSEL2clbkAnimate is not overloaded by the module, RegisterAnimation has no effect.

The RegisterAnimation mechanism leaves the actual implementation of the animation (transformation of mesh groups, etc.) entirely to the module. The [VESSEL::CreateAnimation](#) / [VESSEL::AddAnimationComponent](#) mechanism is an alternative way to define animations where the transformations are managed by the Orbiter core.

**See also:**

[VESSEL2::clbkAnimate](#), [UnregisterAnimation](#), [CreateAnimation](#), [AddAnimationComponent](#)

**16.51.3.243 void VESSEL::UnregisterAnimation () const**

Unlogs an animation request.

**Note:**

This stops a request for animation callback calls from a previous [RegisterAnimation](#).

The call to UnregisterAnimation should not be placed in the body of [VESSEL2::clbkAnimate](#), since it may be lost if the vessel's visual doesn't exist.

**See also:**

[RegisterAnimation](#), [VESSEL2::clbkAnimate](#)

**16.51.3.244 `UINT VESSEL::CreateAnimation (double initial_state) const`**

Create a mesh animation object.

The sequence can contain multiple components (rotations, translations, scalings of mesh groups) with a fixed temporal correlation. The animation is driven by manipulating its "state", which is a number between 0 and 1 used to linearly interpolate the animation within its range. See API User's Guide for details.

**Parameters:**

*initial\_state* the animation state corresponding to the unmodified mesh

**Returns:**

Animation identifier

**Note:**

After creating an animation, components can be added with [AddAnimationComponent](#).

Use [SetAnimation\(\)](#) to manipulate the animation state.

$0 \leq \text{initial\_state} \leq 1$  defines at which state the animation is stored in the mesh file. Example: Landing gear animation between retracted state (0) and deployed state (1). If the landing gear is retracted in the mesh file, set *initial\_state* to 0. If it is deployed in the mesh file, set *initial\_state* to 1.

**See also:**

[DelAnimation](#), [AddAnimationComponent](#)

**16.51.3.245 `bool VESSEL::DelAnimation (UINT anim) const`**

Delete an existing mesh animation object.

**Parameters:**

*anim* animation identifier, as returned by CreateAnimation

**Returns:**

true if animation was deleted successfully

**Note:**

The animation is deleted by removing all the components associated with it. Subsequently, any calls to SetAnimation using this animation index will not have any effect.

Before the animation is deleted, the mesh groups associated with the animation are reset to their default (initial) positions. To avoid jumps in the visual appearance of the vessel, animations should therefore only be deleted when the animation state has returned to the default state.

**See also:**

[CreateAnimation](#)

**16.51.3.246 `ANIMATIONCOMPONENT_HANDLE VESSEL::AddAnimationComponent (UINT anim, double state0, double state1, MGROUP_TRANSFORM * trans, ANIMATIONCOMPONENT_HANDLE parent = NULL) const`**

Add a component (rotation, translation or scaling) to an animation.

Optionally, animations can be stacked hierachically, where transforming a parent recursively also transforms all its children (e.g. a wheel spinning while the landing gear is being retracted).

**Parameters:**

*anim* animation identifier, as returned by [CreateAnimation\(\)](#)  
*state0* animation cutoff state 0 for the component  
*state1* animation cutoff state 1 for the component  
*trans* transformation data (see notes)  
*parent* parent transformation

**Returns:**

Animation component handle

**Note:**

*state0* and *state1* (0..1) allow to define the temporal endpoints of the component's animation within the sequence. For example, *state0*=0 and *state1*=1 perform the animation over the whole duration of the animation sequence, while *state0*=0 and *state1*=0.5 perform the animation over the first half of the total animation. This allows to build complex animations where different components are animated in a defined temporal sequence.

**MGROUP\_TRANSFORM** is the base class for mesh group transforms. Derived classes provide support for rotations, translations and scaling.

To animate a complete mesh, rather than individual mesh groups, set the "grp" pointer to NULL in the constructor of the corresponding **MGROUP\_TRANSFORM** operator. The "ngrp" value is then ignored.

To define a transformation as a child of another transformation, set *parent* to the handle returned by the [AddAnimationComponent](#) call for the parent.

Instead of adding mesh groups to an animation, it is also possible to add a local **VECTOR3** array. To do this, set "mesh" to **LOCALVERTEXLIST**, and set "grp" to **MAKEGROUPARRAY(vtxptr)**, where *vtxptr* is the **VECTOR3** array. "ngrp" is set to the number of vertices in the array. Example:

```
VECTOR3 vtx[2] = {_V(0,0,0), _V(1,0,-1)};
MGROUP_TRANSFORM *mt = new MGROUP_TRANSFORM (LOCALVERTEXLIST,
    MAKEGROUPARRAY(vtx), 2);
AddAnimationComponent (anim, 0, 1, mt);
```

Transforming local vertices in this way does not have an effect on the visual appearance of the animation, but it can be used by the module to keep track of a transformed point during animation. The Atlantis module uses this method to track a grappled satellite during animation of the RMS arm.

The **ANIMATIONCOMPONENT\_HANDLE** is a pointer to a **ANIMATIONCOMP** structure.

**Bug**

When defining a scaling transformation as a child of a parent rotation, only homogeneous scaling is supported, i.e. *scale.x* = *scale.y* = *scale.z* is required.

**See also:**

[CreateAnimation](#), [DelAnimationComponent](#), [Animation flags](#)

**16.51.3.247 bool VESSEL::DelAnimationComponent (UINT anim, ANIMATIONCOMPONENT\_HANDLE hac)**

Remove a component from an animation.

**Parameters:**

*anim* animation identifier

*hAC* animation component handle

**Returns:**

*false* indicates failure (*anim* out of range, or *hAC* invalid)

**Note:**

If the component has children belonging to the same animation, these will be deleted as well.  
In the current implementation, the component must not have children belonging to other animations.  
Trying to delete such a component will result in undefined behaviour.

**See also:**

[AddAnimationComponent](#)

### 16.51.3.248 bool VESSEL::SetAnimation (UINT *anim*, double *state*) const

Set the state of an animation.

**Parameters:**

*anim* animation identifier

*state* animation state (0 ... 1)

**Returns:**

*false* indicates failure (animation identifier out of range)

**Note:**

Each animation is defined by its state, with extreme points state=0 and state=1. When setting a state between 0 and 1, Orbiter carries out the appropriate transformations to advance the animation to that state. It is the responsibility of the code developer to call SetAnimation in such a way as to provide a smooth movement of the animated parts.

### 16.51.3.249 UINT VESSEL::GetAnimPtr (ANIMATION \*\* *anim*) const

Returns a pointer to the array of animations defined by the vessel.

**Parameters:**

→ *anim* pointer list of vessel animations

**Returns:**

list length (number of animations)

**Note:**

The pointer can become invalid whenever the vessel adds or deletes animations. It should therefore not be stored, but queried on demand.

**16.51.3.250 bool VESSEL::Recording () const**

Flag for active recording session.

**Returns:**

*true* if flight recording is active, *false* otherwise.

**See also:**

[Playback](#), [RecordEvent](#)

**16.51.3.251 bool VESSEL::Playback () const**

Flag for active playback session.

**Returns:**

*true* if the current session is a playback of a recorded flight, *false* otherwise.

**See also:**

[Recording](#)

**16.51.3.252 void VESSEL::RecordEvent (const char \* *event\_type*, const char \* *event*) const**

Writes a custom tag to the vessel's articulation data stream during a running recording session.

**Parameters:**

*event\_type* event tag label

*event* event string

**Note:**

This function can be used to record custom vessel events (e.g. animations) to the articulation stream (.atc) of a vessel record.

The function does nothing if no recording is active, so it is not necessary to check for a running recording before invoking RecordEvent.

To read the recorded articulation tags during the playback of a recorded session, overload the [VESSEL2::clbkPlaybackEvent](#) callback function.

**See also:**

[Recording](#), [VESSEL2::clbkPlaybackEvent](#)

**16.51.3.253 void VESSEL::ShiftCentreOfMass (const VECTOR3 & *shift*)**

Register a shift in the centre of mass after a structural change (e.g. stage separation).

**Parameters:**

*shift* centre of mass displacement vector [**m**]

**Note:**

This function should be called after a vessel has undergone a structural change which resulted in a shift of the vessel's centre of gravity (CG). Note that in Orbiter, a vessel's CG coincides by definition always with the origin (0,0,0) of its local reference frame. Therefore, in order to achieve a shift of the CG by a vector  $\mathbf{S}$ , this function shifts the vessel's global position by  $+\mathbf{S}$ . This allows to shift the meshes by  $-\mathbf{S}$ , thus retaining their global position. The net result is unchanged mesh positions in the global frame, but a shift of the local frame of reference (and thus CG) of  $+\mathbf{S}$ .

The camera position is shifted to take into account the new CG. An external camera view performs a smooth transition.

The shift of meshes (and any other reference positions defined in the local vessel frame, such as docking ports, etc.) is not performed by this function but must be executed separately. A more convenient way to implement a transition of the centre of mass is the function [ShiftCG](#), which automatically takes care of translating meshes, docking ports, etc.

**See also:**

[ShiftCG](#)

**16.51.3.254 void VESSEL::ShiftCG (const VECTOR3 & *shift*)**

Shift the centre of gravity of a vessel.

**Parameters:**

*shift* centre of gravity displacement vector [m]

**Note:**

This function should be called after a vessel has undergone a structural change which resulted in a shift of the vessel's centre of gravity (CG). Note that in Orbiter, a vessel's CG coincides by definition always with the origin (0,0,0) of its local reference frame. Therefore, in order to achieve a shift of the CG by *shift*, this function performs the following actions:

- Calls [ShiftCentreOfMass](#) (+*shift*) to align the vessel's global position with the new CG position.
- Calls [ShiftMeshes](#) (-*shift*) to compensate the mesh positions
- Applies equivalent shift to all
  - thruster positions,
  - docking ports,
  - attachment points,
  - explicitly defined light source positions,
  - and to the cockpit camera position

The net effect is a shift of the vessel frame of reference (and thus the CG by +*shift*, while the mesh positions remain in place in the global frame).

**See also:**

[ShiftCentreOfMass](#), [ShiftMeshes](#)

**16.51.3.255 bool VESSEL::GetSuperstructureCG (VECTOR3 & *cg*) const**

Returns the centre of gravity of the superstructure to which the vessel belongs, if applicable.

**Parameters:**

*cg* superstructure centre of gravity [m]

**Returns:**

*true* if the vessel is part of a superstructure, *false* otherwise.

**Note:**

The returned vector is the position of the superstructure centre of gravity, in coordinates of the local vessel frame.

If the vessel is not part of a superstructure, cg returns (0,0,0).

**16.51.3.256 void VESSEL::GetRotationMatrix (MATRIX3 & R) const**

Returns the current rotation matrix for transformations from the vessel's local frame of reference to the global frame.

**Parameters:**

*R* rotation matrix

**Note:**

To transform a point rlocal from local vessel coordinates to a global point rglobal, the following formula is used:

$r_{global} = R r_{local} + p_{vessel}$ ,

where *p<sub>vessel</sub>* is the vessel's global position.

This transformation can be directly performed by a call to Local2Global.

**See also:**

[Local2Global](#), [SetRotationMatrix](#), [GlobalRot](#)

**16.51.3.257 void VESSEL::SetRotationMatrix (const MATRIX3 & R) const**

Applies a rotation by replacing the vessel's local to global rotation matrix.

**Parameters:**

*R* rotation matrix

**Note:**

The rotation matrix maps from the orientation of the vessel's local frame of reference to the orientation of the global frame (ecliptic at 2000.0).

The user is responsible for providing a valid rotation matrix. The matrix must be orthogonal and normalised: the norms of all column vectors of R must be 1, and scalar products between any column vectors must be 0.

**See also:**

[GetRotationMatrix](#), [Local2Global](#)

**16.51.3.258 void VESSEL::GlobalRot (const VECTOR3 & rloc, VECTOR3 & rglob) const**

Performs a rotation of a direction from the local vessel frame to the global frame.

**Parameters:**

- ← **rloc** point in local vessel coordinates
- **rglob** rotated point

**Note:**

This function is equivalent to multiplying rloc with the rotation matrix returned by [GetRotationMatrix](#). Should be used to transform *directions*. To transform *points*, use [Local2Global](#), which additionally adds the vessel's global position to the rotated point.

**See also:**

[GetRotationMatrix](#), [Local2Global](#)

**16.51.3.259 void VESSEL::HorizonRot (const VECTOR3 & rloc, VECTOR3 & rhizon) const**

Performs a rotation from the local vessel frame to the current local horizon frame.

**Parameters:**

- ← **rloc** vector in local vessel coordinates
- **rhizon** vector in local horizon coordinates

**Note:**

The local horizon frame is defined as follows:

- y is "up" direction (planet centre to vessel centre)
- z is "north" direction
- x is "east" direction

**See also:**

[HorizonInvRot](#), [GlobalRot](#), [GetRotationMatrix](#), [SetRotationMatrix](#)

**16.51.3.260 void VESSEL::HorizonInvRot (const VECTOR3 & rhizon, VECTOR3 & rloc) const**

Performs a rotation of a direction from the current local horizon frame to the local vessel frame.

**Parameters:**

- ← **rhizon** vector in local horizon coordinates
- **rloc** vector in local vessel coordinates

**Note:**

This function performs the inverse operation of [HorizonRot](#).

**See also:**

[HorizonRot](#), [GlobalRot](#), [GetRotationMatrix](#), [SetRotationMatrix](#)

**16.51.3.261 void VESSEL::Local2Global (const VECTOR3 & local, VECTOR3 & global) const**

Performs a transformation from local vessel coordinates to global coordinates.

**Parameters:**

- ← *local* point in local vessel coordinates [m]
- *global* transformed point in global coordinates [m]

**Note:**

This function maps a point from the vessel's local coordinate system (centered at the vessel CG) into the global ecliptic system (centered at the solar system barycentre).

The transform has the form

$$\vec{p}_g = R_v \vec{p}_l + \vec{p}_v$$

where  $R_v$  is the vessel's global rotation matrix (as given by [GetRotationMatrix](#)), and  $\vec{p}_v$  is the vessel position in the global frame.

**See also:**

[GetRotationMatrix](#), [Global2Local](#)

**16.51.3.262 void VESSEL::Global2Local (const VECTOR3 & global, VECTOR3 & local) const**

Performs a transformation from global to local vessel coordinates.

**Parameters:**

- ← *global* point in global coordinates [m]
- *local* transformed point in local vessel coordinates [m]

**Note:**

This is the inverse transform of [Local2Global](#). It maps a point from global ecliptic coordinates into the vessel's local frame.

The transformation has the form

$$\vec{p}_l = R_v^{-1} (\vec{p}_g - \vec{p}_v)$$

where  $R_v$  is the vessel's global rotation matrix (as given by [GetRotationMatrix](#)), and  $\vec{p}_v$  is the vessel position in the global frame.

**See also:**

[GetRotationMatrix](#), [Local2Global](#)

**16.51.3.263 void VESSEL::Local2Rel (const VECTOR3 & local, VECTOR3 & rel) const**

Performs a transformation from local vessel coordinates to the ecliptic frame centered at the vessel's reference body.

**Parameters:**

- ← *local* point in local vessel coordinates [m]
- *rel* transformed point in reference body-relative ecliptic coordinates [m].

**Note:**

This function maps a point from the vessel's local coordinate system into an ecliptic system centered at the centre of mass of the vessel's *gravity reference object* (the celestial body that is currently being orbited).

A handle to the reference object can be obtained via [GetGravityRef](#). The reference object may change if the vessel enters a different object's sphere of influence.

The transformation has the form

$$\vec{p}_r = \mathbf{R}_v \vec{p}_l + \vec{p}_v - \vec{p}_{\text{ref}}$$

where  $\mathbf{R}_v$  is the vessel's global rotation matrix (as given by [GetRotationMatrix](#)),  $\vec{p}_v$  is the vessel's global position, and  $\vec{p}_{\text{ref}}$  is the reference body's global position.

**See also:**

[GetRotationMatrix](#), [Global2Local](#), [Local2Global](#), [GetGravityRef](#)

**16.51.3.264 DOCKHANDLE VESSEL::CreateDock (const VECTOR3 & pos, const VECTOR3 & dir, const VECTOR3 & rot) const**

Create a new docking port.

**Parameters:**

*pos* dock reference position in vessel coordinates [m]

*dir* approach direction in vessel coordinates.

*rot* longitudinal rotation alignment vector

**Returns:**

Handle for the new docking port.

**Note:**

The *dir* and *rot* vectors should be normalised to length 1.

The *rot* vector should be perpendicular to the *dir* vector.

When two vessels connect at their docking ports, the relative orientation of the vessels is defined such that their respective approach direction vectors (*dir*) are anti-parallel, and their longitudinal alignment vectors (*rot*) are parallel.

**See also:**

[DelDock](#), [ClearDockDefinitions](#), [GetDockParams](#), [SetDockParams](#), [DockCount](#), [GetDockHandle](#), [GetDockStatus](#), [Dock](#), [Undock](#)

**16.51.3.265 bool VESSEL::DelDock (DOCKHANDLE hDock) const**

Delete a previously defined docking port.

**Parameters:**

*hDock* dock handle

**Returns:**

*false* indicates failure (invalid dock handle)

**Note:**

Any object docked at the port will be undocked before the docking port is deleted.

**See also:**

[CreateDock](#), [ClearDockDefinitions](#), [DockCount](#), [Dock](#), [Undock](#)

**16.51.3.266 void VESSEL::ClearDockDefinitions () const**

Delete all docking ports defined for the vessel.

**Note:**

Any docked objects will be undocked before deleting the docking ports.

**See also:**

[CreateDock](#), [DelDock](#), [DockCount](#), [Dock](#), [Undock](#)

**16.51.3.267 void VESSEL::SetDockParams (const VECTOR3 & pos, const VECTOR3 & dir, const VECTOR3 & rot) const**

Set the parameters for the vessel's primary docking port (port 0), or create a new dock if required.

**Parameters:**

*pos* dock reference position in vessel coordinates [m]

*dir* approach direction in vessel coordinates

*rot* longitudinal rotation alignment vector

**Note:**

This function creates a new docking port if none was previously defined.

See [CreateDock](#) for additional notes on the parameters.

**See also:**

[SetDockParams\(DOCKHANDLE,const VECTOR3&,const VECTOR3&,const VECTOR3&\)const](#),  
[GetDockParams](#), [CreateDock](#), [DelDock](#), [DockCount](#), [Dock](#), [Undock](#)

**16.51.3.268 void VESSEL::SetDockParams (DOCKHANDLE *hDock*, const VECTOR3 & pos, const VECTOR3 & dir, const VECTOR3 & rot) const**

Reset the parameters for a vessel docking port.

**Parameters:**

*hDock* dock handle

*pos* new dock reference position [m]

*dir* new approach direction

*rot* new longitudinal rotation alignment vector

**Note:**

This function should not be called while the docking port is engaged.  
The *dir* and *rot* direction vectors should be normalised to length 1.

**See also:**

[SetDockParams\(const VECTOR3&,const VECTOR3&,const VECTOR3&\)](#)[const](#), [GetDockParams](#),  
[CreateDock](#), [DelDock](#), [DockCount](#), [Dock](#), [Undock](#)

**16.51.3.269 void VESSEL::GetDockParams (DOCKHANDLE *hDock*, VECTOR3 & *pos*, VECTOR3 & *dir*, VECTOR3 & *rot*) const**

Returns the parameters of a docking port.

**Parameters:**

- ← *hDock* dock handle
- *pos* dock reference position [m]
- *dir* approach direction
- *rot* longitudinal rotation alignment vector

**See also:**

[CreateDock](#), [SetDockParams\(const VECTOR3&,const VECTOR3&,const VECTOR3&\)](#)[const](#), [SetDockParams\(DOCKHANDLE,const VECTOR3&,const VECTOR3&,const VECTOR3&\)](#)[const](#)

**16.51.3.270 UINT VESSEL::DockCount () const**

Returns the number of docking ports defined for the vessel.

**Returns:**

Number of docking ports.

**See also:**

[CreateDock](#), [DelDock](#), [ClearDockDefinitions](#)

**16.51.3.271 DOCKHANDLE VESSEL::GetDockHandle (UINT *n*) const**

Returns a handle to a docking port.

**Parameters:**

- n* docking port index ( $\geq 0$ )

**Returns:**

Dock handle, or NULL if index is out of range.

**See also:**

[CreateDock](#), [DelDock](#), [SetDockParams](#), [GetDockParams](#), [GetDockStatus](#), [oapiGetDockHandle](#)

**16.51.3.272 OBJHANDLE VESSEL::GetDockStatus (DOCKHANDLE *hDock*) const**

Returns a handle to a docked vessel.

**Parameters:**

*hDock* dock handle

**Returns:**

Handle of the vessel docked at the specified port, or NULL if the docking port is not engaged.

**See also:**

[CreateDock](#), [GetDockHandle](#), [Dock](#), [Undock](#), [oapiGetDockStatus](#)

**16.51.3.273 UINT VESSEL::DockingStatus (UINT *port*) const**

Returns a status flag for a docking port.

**Parameters:**

*port* docking port index ( $\geq 0$ )

**Returns:**

Docking status (0=free, 1=engaged)

**Note:**

This method has the same functionality as

```
(GetDockStatus (GetDockHandle (port)) ? 1:0)
```

**See also:**

[GetDockStatus](#), [GetDockHandle](#)

**16.51.3.274 int VESSEL::Dock (OBJHANDLE *target*, UINT *n*, UINT *tgn*, UINT *mode*) const**

Dock to another vessel.

**Parameters:**

*target* handle of docking target vessel

*n* docking port index on vessel ( $\geq 0$ )

*tgn* docking port index on target ( $\geq 0$ )

*mode* attachment mode (see notes)

**Returns:**

- 0=ok
- 1=docking port *n* on the vessel already in use
- 2=docking port *tgn* on the target already in use
- 3=target vessel already part of the vessel's superstructure

**Note:**

This function is useful for designing scenario editors and during startup configuration, but its use should be avoided during a running simulation, because it can lead to unphysical situations: it allows to dock two vessels regardless of their current separation, by teleporting one of them to the location of the other.

During a simulation, Orbiter will dock two vessels automatically when their docking ports are brought into close proximity.

The mode parameter determines how the vessels are connected. The following settings are supported:

- 0: calculate the linear and angular moments of the superstructure from the moments of the docking components. This should only be used if the two vessels are already in close proximity and aligned for docking.
- 1: Keep this in place, and teleport the target vessel for docking
- 2: Keep the target in place, and teleport this for docking.

**See also:**

[Undock](#), [GetDockHandle](#), [GetDockStatus](#), [DockCount](#)

**16.51.3.275 bool VESSEL::Undock (UINT *n*, const OBJHANDLE *exclude* = 0) const**

Release a docked vessel from a docking port.

**Parameters:**

*n* docking port index ( $\geq 0$  or ALLDOCKS)

*exclude* optional handle of a vessel to be excluded from undocking

**Returns:**

*true* if at least one vessel was released from a port.

**Note:**

If *n* is set to ALLDOCKS, all docking ports are released simultaneously.

If *exclude* is nonzero, this vessel will not be undocked. This is useful for implementing remote undocking in combination with ALLDOCKS.

**See also:**

[Dock](#), [GetDockHandle](#), [GetDockStatus](#), [DockCount](#)

**16.51.3.276 void VESSEL::SetDockMode (int *mode*) const**

Set the docking approach mode for all docking ports.

**Parameters:**

*mode* docking mode (see notes)

**Note:**

Defines the method Orbiter applies to establish a docking connection between two vessels. Supported values are:

- 0: use legacy (2006) method: snap to dock as soon as two docking ports are within 0.5m and closing.
- 1 (default): use new (2010) method: snap to dock as soon as one docking reference point passes through the reference plane of the other dock within 0.5m.

If the two docking vessels use different docking modes, the method used is unpredictable, depending on which vessel initiates the docking event.

### **16.51.3.277 ATTACHMENTHANDLE VESSEL::CreateAttachment (bool *toparent*, const VECTOR3 & *pos*, const VECTOR3 & *dir*, const VECTOR3 & *rot*, const char \* *id*, bool *loose* = false) const**

Define a new attachment point for a vessel.

**Parameters:**

- toparent*** If *true*, the attachment can be used to connect to a parent (i.e. the vessel acts as a child). Otherwise, attachment is used to connect to a child (i.e. vessel acts as parent)
- pos*** attachment point position in vessel coordinates [**m**]
- dir*** attachment direction in vessel coordinates
- rot*** longitudinal alignment vector in vessel coordinates
- id*** compatibility identifier
- loose*** If *true*, allow loose connections (see notes)

**Returns:**

Handle to new attachment point

**Note:**

A vessel can define multiple parent and child attachment points, and can subsequently have multiple children attached, but it can only be attached to a single parent at any one time.  
The *dir* and *rot* vectors should both be normalised to length 1, and they should be orthogonal.  
The identifier string can contain up to 8 characters. It can be used to define compatibility between attachment points.  
If the attachment point is defined as *loose*, then the relative orientation between the two attached objects is frozen to the orientation between them at the time the connection was established. Otherwise, the two objects snap to the orientation defined by their *dir* vectors.

**See also:**

[SetAttachmentParams](#), [GetAttachmentParams](#), [GetAttachmentId](#), [GetAttachmentStatus](#), [AttachmentCount](#), [GetAttachmentIndex](#), [GetAttachmentHandle](#), [AttachChild](#), [DetachChild](#)

### **16.51.3.278 bool VESSEL::DelAttachment (ATTACHMENTHANDLE *attachment*) const**

Delete an attachment point.

**Parameters:**

***attachment*** attachment handle

**Returns:**

*false* indicates failure (invalid attachment handle)

**Note:**

The attachment handle can refer to either a child or parent attachment point.

Any object attached to this point will be released.

After this function returns successfully, the attachment handle is no longer valid.

**See also:**

[CreateAttachment](#)

**16.51.3.279 void VESSEL::ClearAttachments () const**

Delete all attachment points defined for the vessel.

**Note:**

Any attached parent or child vessels will be released.

After this function returns, all previously defined attachment handles will no longer be valid.

**16.51.3.280 void VESSEL::SetAttachmentParams (ATTACHMENTHANDLE *attachment*, const VECTOR3 & *pos*, const VECTOR3 & *dir*, const VECTOR3 & *rot*) const**

Reset attachment position and orientation for an existing attachment point.

**Parameters:**

*attachment* attachment handle

*pos* new attachment point position in vessel coordinates [**m**]

*dir* new attachment direction in vessel coordinates

*rot* new longitudinal alignment vector in vessel coordinates

**Note:**

If the parameters of an attachment point are changed while a vessel is attached to that point, the attached vessel will be shifted to the new position automatically.

The *dir* and *rot* vectors should both be normalised to length 1, and they should be orthogonal.

**See also:**

[CreateAttachment](#), [GetAttachmentParams](#), [GetAttachmentId](#), [GetAttachmentStatus](#), [AttachmentCount](#), [GetAttachmentIndex](#), [GetAttachmentHandle](#), [AttachChild](#), [DetachChild](#)

**16.51.3.281 void VESSEL::GetAttachmentParams (ATTACHMENTHANDLE *attachment*, VECTOR3 & *pos*, VECTOR3 & *dir*, VECTOR3 & *rot*) const**

Retrieve the parameters of an attachment point.

**Parameters:**

$\leftarrow$  *attachment* attachment handle

- *pos* attachment point position in vessel coordinates [m]
- *dir* attachment direction in vessel coordinates
- *rot* longitudinal alignment vector in vessel coordinates

See also:

[CreateAttachment](#), [SetAttachmentParams](#), [GetAttachmentId](#), [GetAttachmentStatus](#), [AttachmentCount](#), [GetAttachmentIndex](#), [GetAttachmentHandle](#), [AttachChild](#), [DetachChild](#)

#### 16.51.3.282 const char\* VESSEL::GetAttachmentId (ATTACHMENTHANDLE *attachment*) const

Retrieve attachment identifier string.

Parameters:

*attachment* attachment handle

Returns:

Pointer to attachment string [8 characters]

See also:

[CreateAttachment](#), [SetAttachmentParams](#), [GetAttachmentParams](#), [GetAttachmentStatus](#), [AttachmentCount](#), [GetAttachmentIndex](#), [GetAttachmentHandle](#), [AttachChild](#), [DetachChild](#)

#### 16.51.3.283 OBJHANDLE VESSEL::GetAttachmentStatus (ATTACHMENTHANDLE *attachment*) const

Return the current status of an attachment point.

Parameters:

*attachment* attachment handle

Returns:

Handle of the attached vessel, or NULL if no vessel is attached to this point.

See also:

[CreateAttachment](#), [SetAttachmentParams](#), [GetAttachmentParams](#), [GetAttachmentId](#), [AttachmentCount](#), [GetAttachmentIndex](#), [GetAttachmentHandle](#), [AttachChild](#), [DetachChild](#)

#### 16.51.3.284 DWORD VESSEL::AttachmentCount (bool *toparent*) const

Return the number of child or parent attachment points defined for the vessel.

Parameters:

*toparent* If *true*, return the number of attachment points to parents. Otherwise, return the number of attachment points to children.

**Returns:**

Number of defined attachment points to connect to parents or to children.

**See also:**

[CreateAttachment](#), [SetAttachmentParams](#), [GetAttachmentParams](#), [GetAttachmentId](#), [GetAttachmentStatus](#), [GetAttachmentIndex](#), [GetAttachmentHandle](#), [AttachChild](#), [DetachChild](#)

**16.51.3.285 DWORD VESSEL::GetAttachmentIndex (ATTACHMENTHANDLE *attachment*) const**

Return the list index of the vessel's attachment point defined by its handle.

**Parameters:**

*attachment* attachment handle

**Returns:**

List index ( $\geq 0$ )

**Note:**

A vessel defines separate lists for child and parent attachment points. Therefore two different attachment points may return the same index.

The index for a given attachment point can change when the vessel deletes any of its attachments. The returned index should therefore be used only within the current frame.

**See also:**

[CreateAttachment](#), [SetAttachmentParams](#), [GetAttachmentParams](#), [GetAttachmentId](#), [GetAttachmentStatus](#), [AttachmentCount](#), [GetAttachmentHandle](#), [AttachChild](#), [DetachChild](#)

**16.51.3.286 ATTACHMENTHANDLE VESSEL::GetAttachmentHandle (bool *toparent*, DWORD *i*) const**

Return the handle of an attachment point identified by its list index.

**Parameters:**

*toparent* If *true*, return a handle for an attachment point to a parent. Otherwise, return a handle for an attachment point to a child.

*i* attachment index ( $\geq 0$ )

**Returns:**

Attachment handle, or NULL if index out of range.

**See also:**

[CreateAttachment](#), [SetAttachmentParams](#), [GetAttachmentParams](#), [GetAttachmentId](#), [GetAttachmentStatus](#), [AttachmentCount](#), [GetAttachmentIndex](#), [AttachChild](#), [DetachChild](#)

**16.51.3.287 bool VESSEL::AttachChild (OBJHANDLE *child*, ATTACHMENTHANDLE *attachment*, ATTACHMENTHANDLE *child\_attachment*) const**

Attach a child vessel to an attachment point.

**Parameters:**

*child* handle of child vessel to be attached.

*attachment* attachment point to which the child will be attached.

*child\_attachment* attachment point on the child to which we want to attach.

**Returns:**

*true* indicates success, *false* indicates failure (child refuses attachment)

**Note:**

The *attachment* handle must refer to an attachment "to child" (i.e. created with *toparent=false*); the *child\_attachment* handle must refer to an attachment "to parent" on the child object (i.e. created with *toparent=true*). It is not possible to connect two parent or two child attachment points.

A child can only be connected to a single parent at any one time. If the child is already connected to a parent, the previous parent connection is severed.

The child may check the parent attachment's id string and, depending on the value, refuse to connect. In that case, the function returns *false*.

**See also:**

[CreateAttachment](#), [SetAttachmentParams](#), [GetAttachmentParams](#), [GetAttachmentId](#), [GetAttachmentStatus](#), [AttachmentCount](#), [GetAttachmentIndex](#), [GetAttachmentHandle](#), [DetachChild](#)

**16.51.3.288 bool VESSEL::DetachChild (ATTACHMENTHANDLE *attachment*, double *vel* = 0.0) const**

Break an existing attachment to a child.

**Parameters:**

*attachment* attachment handle

*vel* separation velocity [m/s]

**Returns:**

*true* when detachment is successful, *false* if no child was attached, or if child refuses to detach.

**See also:**

[CreateAttachment](#), [SetAttachmentParams](#), [GetAttachmentParams](#), [GetAttachmentId](#), [GetAttachmentStatus](#), [AttachmentCount](#), [GetAttachmentIndex](#), [GetAttachmentHandle](#), [AttachChild](#)

**16.51.3.289 UINT VESSEL::AddExhaust (THRUSTER\_HANDLE *th*, double *lscale*, double *wscale*, SURFHANDLE *tex* = 0) const**

Add an exhaust render definition for a thruster.

**Parameters:**

*th* thruster handle  
*lscale* exhaust flame length [m]  
*wscale* exhaust flame width [m]  
*tex* texture handle for custom exhaust flames

**Returns:**

Exhaust identifier

**Note:**

Thrusters defined with [CreateThruster](#) do not by default render exhaust effects, until an exhaust definition has been specified with [AddExhaust](#).

The size of the exhaust flame is automatically scaled by the thrust level.

This version retrieves exhaust reference position and direction directly from the thruster setting, and will therefore automatically reflect any changes caused by [SetThrusterRef](#) and [SetThrusterDir](#).

To use a custom exhaust texture, set *tex* to a surface handle returned by [oapiRegisterExhaustTexture](#). If *tex* == 0, the default texture is used.

**See also:**

[AddExhaust\(THRUSTER\\_HANDLE,double,double,double,SURFHANDLE\)const](#),  
[AddExhaust\(THRUSTER\\_HANDLE,double,double,const VECTOR3&,const VEC-TOR3&,SURFHANDLE\)const](#), [DelExhaust](#), [CreateThruster](#), [SetThrusterRef](#), [SetThrusterDir](#), [SetThrusterLevel](#), [oapiRegisterExhaustTexture](#)

**16.51.3.290 UINT VESSEL::AddExhaust (THRUSTER\_HANDLE *th*, double *lscale*, double *wscale*, double *lofs*, SURFHANDLE *tex* = 0) const**

Add an exhaust render definition for a thruster with additional offset.

**Parameters:**

*th* thruster handle  
*lscale* exhaust flame length [m]  
*wscale* exhaust flame width [m]  
*lofs* longitudinal offset [m]  
*tex* texture handle for custom exhaust flames

**Returns:**

Exhaust identifier

**Note:**

This method allows to add an additional longitudinal offset between thruster position and exhaust.

**See also:**

[AddExhaust\(THRUSTER\\_HANDLE,double,double,double,SURFHANDLE\)const](#),  
[AddExhaust\(THRUSTER\\_HANDLE,double,double,const VECTOR3&,const VEC-TOR3&,SURFHANDLE\)const](#), [DelExhaust](#), [CreateThruster](#), [SetThrusterRef](#), [SetThrusterDir](#), [SetThrusterLevel](#), [oapiRegisterExhaustTexture](#)

**16.51.3.291 UINT VESSEL::AddExhaust (THRUSTER\_HANDLE *th*, double *lscale*, double *wscale*, const VECTOR3 & *pos*, const VECTOR3 & *dir*, SURFHANDLE *tex* = 0) const**

Add an exhaust render definition for a thruster with explicit reference position and direction.

**Parameters:**

*th* thruster handle  
*lscale* exhaust flame length [m]  
*wscale* exhaust flame width [m]  
*pos* reference position in vessel coordinates [m]  
*dir* exhaust direction in vessel coordinates  
*tex* texture handle for custom exhaust flames

**Note:**

This version uses the explicitly provided reference position and direction, rather than using the thruster parameters.

This allows multiple exhaust render definitions to refer to a single thruster definition, e.g. where multiple thrusters have been combined into a single "logical" thruster definition. This technique can be used to simplify the description of thruster groups which are always addressed synchronously.

The exhaust direction should be opposite to the thrust direction of the thruster it refers to.

Exhaust positions and directions are fixed in this version, so they will not react to changes caused by [SetThrusterRef](#) and [SetThrusterDir](#).

To use a custom exhaust texture, set *tex* to a surface handle returned by [oapiRegisterExhaustTexture](#). If *tex* == 0, the default texture is used.

**See also:**

[AddExhaust\(THRUSTER\\_HANDLE,double,double,SURFHANDLE\)const](#),  
[AddExhaust\(THRUSTER\\_HANDLE,double,double,double,SURFHANDLE\)const](#),  
[DelExhaust](#),  
[CreateThruster](#),  
[SetThrusterRef](#),  
[SetThrusterDir](#),  
[SetThrusterLevel](#),  
[oapiRegisterExhaustTexture](#)

**16.51.3.292 UINT VESSEL::AddExhaust (EXHAUSTSPEC \* *spec*)**

Add an exhaust render definition defined by a parameter structure.

**Parameters:**

*spec* exhaust specification

**Returns:**

Exhaust identifier

**Note:**

This method is more versatile than the other AddExhaust versions. It allows dynamic custom control of exhaust level, position and direction, and it can be defined independently of thrusters.

To let the exhaust appearance be automatically controlled by a thruster, set *spec->th* to the thruster handle. The fields *spec->level*, *spec->lpos* and *spec->ldir* can then be set to NULL, to indicate that they should be linked to the thruster parameters.

If *spec->th* == NULL (thruster-independent exhaust definition), then *spec->level*, *spec->lpos* and *spec->ldir* must not be NULL. They must point to variables that continuously define the level, position

and negative direction of the exhaust cone. The variables themselves must persist during the lifetime of the exhaust definition.

An exception is the definition of a constant parameter. For example, if the exhaust position is to be set to a fixed position, set the spec->flags field to EXHAUST\_CONSTANTPOS. In this case, the value pointed to by spec->lpos is copied by Orbiter, and the variable can be discarded after the call to AddExhaust. In a similar fashion, the bit flags EXHAUST\_CONSTDIR and EXHAUST\_CONSTANTLEVEL can be added to indicate fixed direction and exhaust level, respectively.

If the spec->ldir parameter is provided, it must specify the engine thrust direction (= the negative exhaust direction), in contrast to the other AddExhaust functions, which refer to the positive exhaust direction.

spec->lsize and spec->wsize define the length and width of the exhaust flame [m].

spec->lofs defines a longitudinal offset between the reference position and the exhaust flame.

spec->modulate defines the amplitude of a random variation in exhaust level, between 0 (none) and 1 (max).

spec->tex can be used to provide a custom exhaust texture. If spec->tex == NULL, then the default exhaust texture is used.

#### 16.51.3.293 bool VESSEL::DelExhaust (UINT *idx*) const

Removes an exhaust render definition.

**Parameters:**

*idx* exhaust identifier

**Returns:**

*false* if exhaust definition does not exist, *true* otherwise.

**See also:**

[AddExhaust](#), [GetExhaustCount](#)

#### 16.51.3.294 DWORD VESSEL::GetExhaustCount () const

Returns the number of exhaust render definitions for the vessel.

**Returns:**

Number of exhaust render definitions

**See also:**

[AddExhaust](#), [DelExhaust](#)

#### 16.51.3.295 bool VESSEL::GetExhaustSpec (UINT *idx*, double \* *lscale*, double \* *wscale*, VECTOR3 \* *pos*, VECTOR3 \* *dir*, SURFHANDLE \* *tex*) const

Returns the parameters of an exhaust definition.

**Parameters:**

← *idx* exhaust identifier

→ *lscale* exhaust flame length [m]

- **wscale** exhaust flame width [m]
- **pos** reference position [m]
- **dir** exhaust direction
- **tex** texture handle for custom exhaust flames, if any

**Returns:**

*false* if *idx* out of range, *true* otherwise.

**See also:**

[AddExhaust](#)

**16.51.3.296 bool VESSEL::GetExhaustSpec (UINT *idx*, EXHAUSTSPEC \* *spec*)**

Returns the parameters of an exhaust definition in a structure.

**Parameters:**

- ← **idx** exhaust identifier
- **spec** pointer to [EXHAUSTSPEC](#) structure

**Returns:**

*false* if *idx* is out of range, *true* otherwise.

**Note:**

On return the parameters of the specified exhaust object are copied into the structure pointed to by *spec*.

**16.51.3.297 double VESSEL::GetExhaustLevel (UINT *idx*) const**

Returns the current level of an exhaust source.

**Parameters:**

- idx** exhaust identifier

**Returns:**

Exhaust level (0..1)

**Note:**

The exhaust level is equivalent to the thrust level of the thruster to which the exhaust definition is attached.

**See also:**

[AddExhaust](#), [GetThrusterLevel](#)

**16.51.3.298 void VESSEL::SetReentryTexture (SURFHANDLE *tex*, double *plimit* = 6e7, double *lscale* = 1.0, double *wscale* = 1.0) const**

Select a previously registered texture to be used for rendering reentry flames.

**Parameters:**

*tex* texture handle  
*plimit* friction power limit  
*lscale* texture length scaling factor  
*wscale* texture width scaling factor

**Note:**

The texture handle is obtained by a previous call to [oapiRegisterReentryTexture](#).  
If a custom texture is not explicitly set, Orbiter uses a default texture (reentry.dds) for rendering reentry flames. To suppress reentry flames altogether for a vessel, call SetReentryTexture(NULL).

**See also:**

[oapiRegisterReentryTexture](#)

**16.51.3.299 PSTREAM\_HANDLE VESSEL::AddParticleStream (PARTICLESTREAMSPEC \* *pss*, const VECTOR3 & *pos*, const VECTOR3 & *dir*, double \* *lvl*) const**

Adds a custom particle stream to a vessel.

**Parameters:**

*pss* pointer to particle stream definition structure  
*pos* particle source position in vessel coordinates [m]  
*dir* particle emission direction in vessel coordinates  
*lvl* pointer to scaling factor

**Returns:**

Particle stream handle

**Note:**

This function can be used to add venting effects and similar. For engine-specific effects such as exhaust and contrails, use the [AddExhaustStream](#) functions instead.

The [PARTICLESTREAMSPEC](#) structure defined the properties of the particle stream.

The position and direction variables are in vessel-relative coordinates. They cannot be redefined.

*lvl* points to a variable which defines the strength of the particle emission. Its value should be set in the range from 0 (particle generation off) to 1 (emission at full strength). It can be changed continuously to modulate the particle generation.

**See also:**

[AddExhaustStream](#), [AddReentryStream](#)

**16.51.3.300 PSTREAM\_HANDLE VESSEL::AddExhaustStream (THRUSTER\_HANDLE *th*,  
PARTICLESTREAMSPEC \**pss* = 0) const**

Adds an exhaust particle stream to a vessel.

**Parameters:**

*th* thruster handle

*pss* particle stream specification

**Returns:**

Particle stream handle

**Note:**

Exhaust streams can be emissive (to simulate "glowing" ionised gases) or diffuse (e.g. for simulating vapour trails).

The **PARTICLESTREAMSPEC** structure defined the properties of the particle stream.

Multiple streams can be defined for a single engine. For example, an emissive stream with short lifetime may represent the ionised exhaust gases, while a diffuse stream with longer lifetime represents the vapour trail.

To improve performance, closely packed engines may share a single exhaust stream.

If the user has disabled particle streams in the launchpad dialog, this function will return NULL. The module must be able to cope with this case.

**See also:**

[AddExhaustStream\(THRUSTER\\_HANDLE,const VECTOR3&,PARTICLESTREAMSPEC\\*\)const](#),  
[AddParticleStream](#), [AddReentryStream](#)

**16.51.3.301 PSTREAM\_HANDLE VESSEL::AddExhaustStream (THRUSTER\_HANDLE *th*,  
const VECTOR3 &*pos*, PARTICLESTREAMSPEC \**pss* = 0) const**

Adds an exhaust particle stream to a vessel.

**Parameters:**

*th* thruster handle

*pos* particle emission reference point

*pss* particle stream specification

**Returns:**

Particle stream handle

**Note:**

This version allows to pass an explicit particle emission reference position, independent of the engine reference point.

If the user has disabled particle streams in the launchpad dialog, this function will return NULL. The module must be able to cope with this case.

**See also:**

[AddExhaustStream\(THRUSTER\\_HANDLE,PARTICLESTREAMSPEC\\*\)const](#), [AddParticleStream](#),  
[AddReentryStream](#)

**16.51.3.302 PSTREAM\_HANDLE VESSEL::AddReentryStream (PARTICLESTREAMSPEC \* *pss*) const**

Adds a reentry particle stream to a vessel.

**Parameters:**

*pss* particle stream specification

**Returns:**

Particle stream handle

**Note:**

Vessels automatically define a default emissive particle stream, but you may want to add further stream to customise the appearance.

**See also:**

[AddParticleStream](#), [AddExhaustStream](#)

**16.51.3.303 bool VESSEL::DelExhaustStream (PSTREAM\_HANDLE *ch*) const**

Delete an existing particle stream.

**Parameters:**

*ch* particle stream handle

**Returns:**

*false* indicates failure (particle stream not found)

**Note:**

If a thruster is deleted (with ref DelThruster), any attached particle streams are deleted automatically. A deleted particle stream will no longer emit particles, but existing particles persist until they expire.

**See also:**

[AddParticleStream](#), [AddExhaustStream](#), [AddReentryStream](#)

**16.51.3.304 void VESSEL::SetNosewheelSteering (bool *activate*) const****Parameters:**

*activate* *true* to activate, *false* to deactivate

**Note:**

With nose-wheel steering active, the yaw controls will apply a lateral force on the front touchdown-point when in ground contact.

By default, nose-wheel steering is inactive. This function should only be called for appropriate vessel types.

**See also:**

[GetNosewheelSteering](#)

**16.51.3.305 bool VESSEL::GetNosewheelSteering () const**

Returns the activation state of the nose-wheel steering system.

**Returns:**

*true* indicates nose-wheel steering is active, *false* indicates disabled.

**See also:**

[SetNosewheelSteering](#)

**16.51.3.306 void VESSEL::SetMaxWheelbrakeForce (double *f*) const**

Define the maximum force which can be provided by the vessel's wheel brake system.

**Parameters:**

*f* maximum force [N]

**See also:**

[SetWheelbrakeLevel](#), [GetWheelbrakeLevel](#)

**16.51.3.307 void VESSEL::SetWheelbrakeLevel (double *level*, int *which* = 0, bool *permanent* = true) const**

Apply the wheel brake.

**Parameters:**

*level* wheelbrake level [0..1]

*which* 0 = both, 1 = left, 2 = right main gear

*permanent* *true* sets the level permanently, *false* only applies to current time step

**See also:**

[SetMaxWheelbrakeForce](#), [GetWheelbrakeLevel](#)

**16.51.3.308 double VESSEL::GetWheelbrakeLevel (int *which*) const**

Returns the current wheel brake level.

**Parameters:**

*which* 0 = average of both main gear levels, 1 = left, 2 = right

**Returns:**

wheel brake level [0..1]

**See also:**

[SetMaxWheelbrakeForce](#), [SetWheelbrakeLevel](#)

**16.51.3.309 void VESSEL::AddBeacon (BEACONLIGHTSPEC \* *bs*)**

Add a light beacon definition to a vessel.

**Parameters:**

*bs* structure defining the beacon parameters

**Note:**

The **BEACONLIGHTSPEC** variable passed to AddBeacon (as well as the pos and col vectors pointed to by the structure) must remain valid until the beacon is removed (with DelBeacon, ClearBeacons, or by deleting the vessel). It should therefore either be defined static, or as a member of the derived vessel class.

The **BEACONLIGHTSPEC** parameters can be modified at any time by the module after the call to AddBeacon, to modify the beacon appearance. The changes take effect immediately.

To turn the beacon off temporarily, don't delete the beacon but simply set the *active* element to false. *shape* defines the appearance of the beacon. Currently supported are:

- BEACONSHAPE\_COMPACT (a compact blob)
- BEACONSHAPE\_DIFFUSE (a more diffuse blob)
- BEACONSHAPE\_STAR (a starlike appearance)

*falloff* determines how the render size of the beacon changes with distance. The value should be between 0 and 1, where 0 means that the apparent size of the beacon is proportional to 1/distance, and 1 means that the apparent size doesn't change at all with distance. The higher the value, the further away the beacon will remain visible. (but note that visibility is limited to the range defined by [SetVisibilityLimit](#)).

*period*, *duration* and *tofs* are used to define a periodically blinking beacon (strobe). To define a continuous beacon, set period = 0. The two other parameters are then ignored.

**See also:**

[DelBeacon](#), [ClearBeacons](#), [SetVisibilityLimit](#)

**16.51.3.310 bool VESSEL::DelBeacon (BEACONLIGHTSPEC \* *bs*)**

Remove a beacon definition from the vessel.

**Parameters:**

*bs* pointer to the **BEACONLIGHTSPEC** structure previously use to define the beacon with AddBeacon.

**Returns:**

*true* if the beacon definition was found and removed, *false* otherwise.

**Note:**

DelBeacon removes the beacon reference from the vessel's list of beacons, but does not deallocate the beacon itself. If the vessel had defined the beacon specification dynamically, it should deallocate it after this call.

**See also:**

[AddBeacon](#), [ClearBeacons](#)

**16.51.3.311 void VESSEL::ClearBeacons ()**

Remove all beacon definitions from the vessel.

**See also:**

[AddBeacon](#), [DelBeacon](#)

**16.51.3.312 const BEACONLIGHTSPEC\* VESSEL::GetBeacon (DWORD idx) const**

Returns a pointer to one of the vessel's beacon specifications.

**Parameters:**

*idx* beacon list index ( $\geq 0$ )

**Returns:**

Pointer to specification for vessel beacon at list index *idx*, or NULL if *idx* is out of range.

**Note:**

The list index for a given beacon can change when the vessel adds or deletes beacons.

**16.51.3.313 LightEmitter\* VESSEL::AddPointLight (const VECTOR3 & pos, double range, double att0, double att1, double att2, COLOUR4 diffuse, COLOUR4 specular, COLOUR4 ambient) const**

\ name Light emitters

Add an isotropic point light source to the vessel.

**Parameters:**

*pos* source position [m] in vessel coordinates

*range* light source range [m]

*att0* attenuation coefficients (see notes)

*att1* attenuation coefficients (see notes)

*att2* attenuation coefficients (see notes)

*diffuse* source contribution to diffuse object colours

*specular* source contribution to specular object colours

*ambient* source contribution to ambient object colours

**Returns:**

pointer to new emitter object

**Note:**

The intensity *I* of the light source as a function of distance *d* is defined via the coefficients by

$$I = \frac{1}{att_0 + datt_1 + d^2att_2}$$

**16.51.3.314 LightEmitter\* VESSEL::AddSpotLight (const VECTOR3 & *pos*, const VECTOR3 & *dir*, double *range*, double *att0*, double *att1*, double *att2*, double *umbra*, double *penumbra*, COLOUR4 *diffuse*, COLOUR4 *specular*, COLOUR4 *ambient*) const**

Add a directed spot light source to the vessel.

**Parameters:**

*pos* source position [m] in vessel coordinates  
*dir* light direction in vessel coordinates  
*range* light source range [m]  
*att0* attenuation coefficients (see notes)  
*att1* attenuation coefficients (see notes)  
*att2* attenuation coefficients (see notes)  
*umbra* aperture of inner (maximum intensity) cone [rad]  
*penumbra* aperture of outer (zero intensity) cone [rad]  
*diffuse* source contribution to diffuse object colours  
*specular* source contribution to specular object colours  
*ambient* source contribution to ambient object colours

**Returns:**

pointer to new emitter object

**Note:**

The intensity  $I$  of the light source as a function of distance  $d$  is defined via the coefficients by

$$I = \frac{1}{att_0 + datt_1 + d^2att_2}$$

**16.51.3.315 DWORD VESSEL::LightEmitterCount () const**

Returns the number of light sources defined for the vessel.

**Returns:**

Number of light sources.

**16.51.3.316 const LightEmitter\* VESSEL::GetLightEmitter (DWORD *i*) const**

Returns a pointer to a light source object identified by index.

**Parameters:**

*i* emitter index ( $\geq 0$ )

**Returns:**

Pointer to light source object, or NULL if index out of range

**Note:**

The index of a given source object can change if other objects in the list are deleted.

**See also:**

[LightEmitterCount](#)

**16.51.3.317 bool VESSEL::DelLightEmitter (LightEmitter \* *le*) const**

Deletes the specified light source from the vessel.

**Parameters:**

*le* pointer to light emitter object

**Returns:**

*true* if the emitter was successfully deleted, *false* if the source was not recognised by the vessel.

**Note:**

If the method returns *true*, the emitter (*le*) was deallocated and the pointer should no longer be used.

**See also:**

[ClearLightEmitters](#), [LightEmitterCount](#)

**16.51.3.318 void VESSEL::ClearLightEmitters () const**

Remove all light sources defined for the vessel.

**See also:**

[AddPointLight](#), [AddSpotLight](#), [LightEmitterCount](#)

**16.51.3.319 void VESSEL::ParseScenarioLineEx (char \* *line*, void \* *status*) const**

Pass a line read from a scenario file to Orbiter for default processing.

**Parameters:**

*line* line to be interpreted

*status* status parameters (points to a VESSELSTATUSx variable).

**Note:**

This function should be used within the body of [VESSEL2::clbkLoadStateEx](#).

The parser clbkLoadStateEx should forward all lines not recognised by the module to Orbiter via ParseScenarioLineEx to allow processing of standard vessel settings.

clbkLoadStateEx currently provides a [VESSELSTATUS2](#) status definition. This may change in future versions, so status should not be used within clbkLoadStateEx other than passing it to ParseScenarioLineEx.

**See also:**

[VESSEL2::clbkLoadStateEx](#)

**16.51.3.320 void VESSEL::SetEngineLevel (ENGINETYPE *eng*, double *level*) const**

Set the thrust level for an engine group.

**Deprecated**

This method has been replaced by [VESSEL::SetThrusterGroupLevel](#).

**Parameters:**

*eng* engine group identifier  
*level* thrust level [0..1]

**See also:**

[SetThrusterGroupLevel](#), [IncEngineLevel](#)

**16.51.3.321 void VESSEL::IncEngineLevel (ENGINETYPE *eng*, double *dlevel*) const**

Increase or decrease the thrust level for an engine group.

**Deprecated**

This method has been replaced by [VESSEL::IncThrusterGroupLevel](#).

**Parameters:**

*eng* engine group identifier  
*dlevel* thrust increment

**Note:**

Use negative dlevel to decrease the engine's thrust level.  
Levels are clipped to valid range.

**See also:**

[IncThrusterGroupLevel](#), [SetEngineLevel](#)

**16.51.3.322 void VESSEL::SetExhaustScales (EXHAUSTTYPE *exh*, WORD *id*, double *lscale*, double *wscale*) const****Deprecated**

This method no longer performs any action. It has been replaced by the [VESSEL::AddExhaust](#) methods.

**See also:**

[AddExhaust\(THRUSTER\\_HANDLE,double,double,SURFHANDLE\)const](#),  
[AddExhaust\(THRUSTER\\_HANDLE,double,double,double,SURFHANDLE\)const](#),  
[AddExhaust\(THRUSTER\\_HANDLE,double,double,const VECTOR3&,const VECTORT3&,SURFHANDLE\)const](#)

**16.51.3.323 bool VESSEL::DelThrusterGroup (THGROUP\_HANDLE & *thg*, THGROUP\_TYPE *thgt*, bool *delth* = false) const**

Delete a thruster group and (optionally) all associated thrusters.

**Deprecated**

This method has been replaced by [VESSEL::DelThrusterGroup\(THGROUP\\_HANDLE,bool\)const](#).

**Parameters:**

*thg* thruster group handle (NULL on return)

*thgt* thruster group type (see [Thruster and thruster-group parameters](#))

*delth* thruster destruction flag (see notes)

**Returns:**

*true* on success.

**Note:**

If *delth==true*, all thrusters associated with the group will be destroyed. Note that this can have side effects if the thrusters were associated with multiple groups, since they are removed from all those groups as well.

**See also:**

[DelThrusterGroup\(THGROUP\\_TYPE,bool\)const](#), [CreateThrusterGroup](#), [DelThruster](#), [Thruster and thruster-group parameters](#)

**16.51.3.324 double VESSEL::GetBankMomentScale () const**

Returns the scaling factor for the yaw moment.

**Deprecated**

This method has been replaced by [VESSEL::GetYawMomentScale](#).

**Returns:**

yaw moment scale factor

**Note:**

The method is misnamed. It refers to the vessel's yaw moment.

**See also:**

[GetYawMomentScale](#)

**16.51.3.325 void VESSEL::SetBankMomentScale (double *scale*) const**

Sets the scaling factor for the yaw moment.

**Deprecated**

This method has been replaced by [VESSEL::SetYawMomentScale](#).

**Parameters:**

*scale* scale factor for slip angle moment.

**Note:**

The method is misnamed. It refers to the vessel's yaw moment.

**See also:**

[SetYawMomentScale](#)

**16.51.3.326 bool VESSEL::SetNavRecv (DWORD *n*, DWORD *ch*) const**

Sets the channel of a NAV radio receiver.

**Deprecated**

This method has been replaced by [VESSEL::SetNavChannel](#)

**Parameters:**

*n* receiver index ( $\geq 0$ )

*ch* channel ( $\geq 0$ )

**Returns:**

*false* on error (index out of range), *true* otherwise

**16.51.3.327 DWORD VESSEL::GetNavRecv (DWORD *n*) const**

Returns the current channel setting of a NAV radio receiver.

**Deprecated**

This method has been replaced by [VESSEL::GetNavChannel](#)

**Parameters:**

*n* receiver index ( $\geq 0$ )

**Returns:**

Receiver channel [0..639]. If index *n* is out of range, the return value is 0.

**16.51.3.328 void VESSEL::SetCOG\_elev (double *h*) const**

Set the altitude of the vessel's centre of gravity over ground level when landed.

**Parameters:**

*h* elevation of the vessel's centre of gravity above the surface plane when landed [m].

**Deprecated**

This method is obsolete and should no longer be used. It has been replaced by [VESSEL::SetTouchdownPoints](#).

**16.51.3.329 void VESSEL::ClearMeshes () const**

Remove all mesh definitions for the vessel.

**Deprecated**

This version is obsolete and has been replaced by [VESSEL::ClearMeshes\(bool\)const](#).

**Note:**

Equivalent to ClearMeshes(true). This method is only retained for backward compatibility, and may be removed in future versions.

**See also:**

[ClearMeshes\(bool\)const](#)

**16.51.3.330 void VESSEL::SetMeshVisibleInternal (UINT *idx*, bool *visible*) const**

Marks a mesh as visible from internal cockpit view.

**Parameters:**

*idx* mesh index ( $\geq 0$ )

*visible* visibility flag

**Deprecated**

This method is obsolete and has been replaced by [VESSEL::SetMeshVisibilityMode](#).

**Note:**

By default, a vessel is not rendered when the camera is in internal (cockpit) view. This function can be used to force rendering of some or all of the vessel's meshes.

**See also:**

[SetMeshVisibilityMode](#)

**16.51.3.331 void VESSEL::SaveDefaultState (FILEHANDLE *scn*) const**

Causes Orbiter to write default vessel parameters to a scenario file.

**Deprecated**

Use a call to the base class [VESSEL2::clbkSaveState](#) from within the overloaded callback function instead.

**Parameters:**

*scn* scenario file handle

**Note:**

This method saves the vessel's default state parameters (such as position, velocity, orientation, etc.) to a scenario file.

This functionality is now included in the default implementation of [VESSEL2::clbkSaveState](#). Therefore, vessel classes which overload this method to save custom vessel parameters should call the base class method to allow Orbiter to save the default vessel parameters.

See also:

[VESSEL2::clbkSaveState](#)

### 16.51.3.332 void VESSEL::ParseScenarioLine (char \* *line*, VESSELSTATUS \* *status*) const

Pass a line read from a scenario file to Orbiter for default processing.

#### Deprecated

This function is retained for backward compatibility only. New modules should overload the [VESSEL2::clbkLoadStateEx](#) function and use [VESSEL::ParseScenarioLineEx](#) for default state parsing.

#### Parameters:

*line* line to be interpreted

*status* state parameter set

See also:

[ParseScenarioLineEx](#), [VESSELSTATUS](#)

### 16.51.3.333 static OBJHANDLE VESSEL::Create (const char \* *name*, const char \* *classname*, const VESSELSTATUS & *status*) [static]

Vessel creation.

#### Deprecated

This method has been replaced with [oapiCreateVessel](#) and [oapiCreateVesselEx](#).

The documentation for this class was generated from the following file:

- Orbitersdk/include/[VesselAPI.h](#)

## 16.52 VESSEL2 Class Reference

```
#include <VesselAPI.h>
```

Inheritance diagram for VESSEL2:

Collaboration diagram for VESSEL2:

### 16.52.1 Detailed Description

Callback extensions to the [VESSEL](#) class.

The [VESSEL2](#) class adds a variety of callback functions to the [VESSEL](#) interface (clbk\*). These are called by Orbiter to notify the vessel about different types of events and allow it to react to them. The [VESSEL2](#) class implements these as virtual functions which act as placeholders to be overwritten by derived classes whenever a non-default behaviour is required.

#### Examples:

[clbkLoadStateEx.cpp](#), [clbkPreStep.cpp](#), [clbkSetStateEx.cpp](#), and [VESSEL2.cpp](#).

#### Public Member Functions

- [VESSEL2 \(OBJHANDLE hVessel, int fmodel=1\)](#)  
*Creates a [VESSEL2](#) interface for a vessel object.*
- virtual void [clbkSetClassCaps \(FILEHANDLE cfg\)](#)  
*Initialisation of vessel capabilities.*
- virtual void [clbkSaveState \(FILEHANDLE scn\)](#)  
*Called when the vessel needs to save its current status to a scenario file.*
- virtual void [clbkLoadStateEx \(FILEHANDLE scn, void \\*status\)](#)  
*Called when the vessel needs to load its initial state from a scenario file.*
- virtual void [clbkSetStateEx \(const void \\*status\)](#)  
*Set state parameters during vessel creation.*
- virtual void [clbkPostCreation \(\)](#)  
*Called after a vessel has been created and its state has been set.*
- virtual void [clbkFocusChanged \(bool getfocus, OBJHANDLE hNewVessel, OBJHANDLE hOldVessel\)](#)  
*Called after a vessel gained or lost input focus.*
- virtual void [clbkPreStep \(double simt, double simdt, double mjd\)](#)  
*Time step notification before state update.*
- virtual void [clbkPostStep \(double simt, double simdt, double mjd\)](#)  
*Time step notification after state update.*

- virtual bool `clbkPlaybackEvent` (double simt, double event\_t, const char \*event\_type, const char \*event)  
*Playback event notification.*
- virtual void `clbkVisualCreated` (**VISHANDLE** vis, int refcount)  
*Called after a vessel visual has been created by the renderer.*
- virtual void `clbkVisualDestroyed` (**VISHANDLE** vis, int refcount)  
*Called before a vessel visual is destroyed.*
- virtual void `clbkDrawHUD` (int mode, const **HUDPAINTSPEC** \*hps, **HDC** hDC)  
*HUD redraw notification.*
- virtual void `clbkRCSMode` (int mode)  
*Reaction Control System mode change notification.*
- virtual void `clbkADCtrlMode` (DWORD mode)  
*Aerodynamic control surface mode change notification.*
- virtual void `clbkHUDMode` (int mode)  
*HUD mode change notification.*
- virtual void `clbkMFDMode` (int mfd, int mode)  
*MFD mode change modification.*
- virtual void `clbkNavMode` (int mode, bool active)  
*Navigation mode change notification.*
- virtual void `clbkDockEvent` (int dock, **OBJHANDLE** mate)  
*Docking event notification.*
- virtual void `clbkAnimate` (double simt)  
*Manual animation notification.*
- virtual int `clbkConsumeDirectKey` (char \*kstate)  
*Keyboard status notification.*
- virtual int `clbkConsumeBufferedKey` (DWORD key, bool down, char \*kstate)  
*Keyboard event notification.*
- virtual bool `clbkLoadGenericCockpit` ()  
*Generic cockpit view mode request notification.*
- virtual bool `clbkLoadPanel` (int id)  
*2-D instrument panel view mode request notification*
- virtual bool `clbkPanelMouseEvent` (int id, int event, int mx, int my)  
*Mouse event notification for 2-D panel views.*

- virtual bool `clbkPanelRedrawEvent` (int id, int event, SURFHANDLE surf)  
*Redraw event notification for 2-D panel views.*
- virtual bool `clbkLoadVC` (int id)  
*3-D virtual cockpit view mode request notification*
- virtual bool `clbkVCMouseEvent` (int id, int event, VECTOR3 &p)  
*Mouse event notification for 3-D virtual cockpit views.*
- virtual bool `clbkVCRedrawEvent` (int id, int event, SURFHANDLE surf)  
*Redraw event notification for 3-D virtual cockpit views.*

### 16.52.2 Constructor & Destructor Documentation

#### 16.52.2.1 VESSEL2::VESSEL2 (OBJHANDLE *hVessel*, int *fmodel* = 1)

Creates a `VESSEL2` interface for a vessel object.

An instance of a vessel class derived from `VESSEL2` is typically called during the initialisation of a vessel module (during `ovcInit`) to create an interface to the vessel instance controlled by the module. However, a `VESSEL2` instance for any existing vessel can be created by any module.

**Parameters:**

*hVessel* vessel object handle  
*fmodel* requested level of realism (0=simple, 1=realistic)

**Note:**

This function creates an interface to an *existing* vessel. It does not create a new vessel. New vessels are created with the `oapiCreateVessel` and `oapiCreateVesselEx` functions.

The `VESSEL2` interface instance created in `ovcInit` should be deleted in `ovcExit`.

**See also:**

`oapiCreateVessel`, `oapiCreateVesselEx`, `ovcInit`

### 16.52.3 Member Function Documentation

#### 16.52.3.1 virtual void VESSEL2::clbkSetClassCaps (FILEHANDLE *cfg*) [virtual]

Initialisation of vessel capabilities.

Called after vessel creation, this function allows to set vessel class capabilities and parameters. This can include definition of physical properties (size, mass, docking ports, etc.), creation of propellant resources and engines, aerodynamic parameters, including airfoil definitions, lift and drag properties, or active control surfaces.

**Parameters:**

*cfg* handle for the vessel class configuration file

**Default action:**

None.

**Note:**

This function is called after the vessel has been created, but before its state is read from the scenario file. This means that its state (position, velocity, fuel level, etc.) is undefined at this point.

Use this function to set vessel class capabilities, not vessel state parameters.

Orbiter will scan the vessel class configuration file for generic parameters (like mass or size) after clbkSetClassCaps returns. This allows to override generic caps defined in the module by editing the configuration file.

The configuration file handle is also passed to clbkSetClassCaps, to allow reading of vessel class-specific parameters from file.

**16.52.3.2 virtual void VESSEL2::clbkSaveState (FILEHANDLE *scn*) [virtual]**

Called when the vessel needs to save its current status to a scenario file.

**Parameters:**

*scn* scenario file handle

**Default action:**

Saves the generic vessel state parameters.

**Note:**

clbkSaveState is called by Orbiter at the end of a simulation session while creating the save scenario for the current simulation state.

This function only needs to be overloaded if the vessel must save nonstandard parameters.

If clbkSaveState is overloaded, generic state parameters will only be written if the base class [VESSEL2::clbkSaveState](#) is called.

To write custom parameters to the scenario file, use the oapiWriteLine function.

**16.52.3.3 void VESSEL2::clbkLoadStateEx (FILEHANDLE *scn*, void \* *status*) [virtual]**

Called when the vessel needs to load its initial state from a scenario file.

**Parameters:**

*scn* scenario file handle

*status* pointer to VESSELSTATUSx structure ( $x \geq 2$ )

**Default action:**

Loads the generic vessel state parameters.

**Note:**

This callback function allows to read custom vessel status parameters from a scenario file.

The function should define a loop which parses lines from the scenario file via oapiReadScenario\_nextline.

You should not call the base class clbkLoadStateEx to parse generic parameters, because this will skip over any custom scenario entries. Instead, any lines which the module parser does not recognise should be forwarded to Orbiter's default scenario parser via [VESSEL::ParseScenarioLineEx](#).

**See also:**

[VESSELSTATUS2](#), [ParseScenarioLineEx](#), [oapiReadScenario\\_nextline](#)

**Examples:**

[clbkLoadStateEx.cpp](#).

**16.52.3.4 void VESSEL2::clbkSetStateEx (const void \* *status*) [virtual]**

Set state parameters during vessel creation.

**Parameters:**

*status* pointer to a VESSELSTATUSx structure

**Default action:**

Invokes Orbiter's default state initialisation.

**Calling sequence:**

This function is called when the vessel is being created with oapiCreateVesselEx, after its clbkSetClassCaps has been invoked and before its clbkPostCreation method is invoked. Vessels that are created during simulation start as a result of parsing the scenario file invoke clbkLoadStateEx instead.

**Note:**

This callback function receives the VESSELSTATUSx structure passed to oapiCreateVesselEx. It must therefore be able to process the interface version used by those functions.

This function remains valid even if future versions of Orbiter introduce new VESSELSTATUSx interfaces.

If an overloaded method does not call [VESSEL2::clbkSetStateEx](#), no default state initialisation is performed. Default state initialisation can also be done by calling [VESSEL::DefSetStateEx](#).

**Examples:**

[clbkSetStateEx.cpp](#).

**16.52.3.5 virtual void VESSEL2::clbkPostCreation () [virtual]**

Called after a vessel has been created and its state has been set.

**Default action:**

None.

**Calling sequence:**

This function is called during vessel creation after clbkSetStateEx or clbkLoadStateEx have been called and before the vessel enters the update loop, i.e. before its clbkPreStep is invoked for the first time. Vessels that are created at the start of the simulation (i.e. are listed in the scenario) call their clbkPostCreation after all scenario vessels have been created.

**Note:**

This function can be used to perform the final setup steps for the vessel, such as animation states and instrument panel states. When this function is called, the vessel state (e.g. position, thruster levels, etc.) have been defined.

**16.52.3.6 virtual void VESSEL2::clbkFocusChanged (bool *getfocus*, OBJHANDLE *hNewVessel*, OBJHANDLE *hOldVessel*) [virtual]**

Called after a vessel gained or lost input focus.

**Parameters:**

*getfocus* true if the vessel gained focus, false if it lost focus

*hNewVessel* handle of vessel gaining focus

*hOldVessel* handle of vessel losing focus

**Default action:**

None.

**Note:**

Whenever the input focus is switched to a new vessel (e.g. via user selection F3), this method is called for both the vessel losing focus (*getfocus*=false) and the vessel gaining focus (*getfocus*=true).

In both calls, *hNewVessel* and *hOldVessel* are the vessel handles for the vessel gaining and the vessel losing focus, respectively.

This method is also called at the beginning of the simulation for the initial focus object. In this case *hOldVessel* is NULL.

**16.52.3.7 void VESSEL2::clbkPreStep (double *simt*, double *simdt*, double *mjd*) [virtual]**

Time step notification before state update.

Called at each simulation time step before the state is updated to the current simulation time. This function allows to define actions which need to be controlled continuously.

**Parameters:**

*simt* next simulation run time [s]

*simdt* step length over which the current state will be integrated [s]

*mjd* next absolute simulation time (days) in Modified Julian Date format

**Default action:**

None

**Note:**

This function is called at each frame of the simulation, after the integration step length has been determined, but before the time integration is applied to the current simulation state.

This method is useful when the step length Dt is required in advance of the time integration, for example to apply a force that produces a given Dv, since the AddForce request will be applied in the next update. Using clbkPostStep for this purpose would be wrong, because its Dt parameter refers to the previous step length.

**See also:**

[clbkPostStep](#)

**Examples:**

[clbkPreStep.cpp](#).

**16.52.3.8 virtual void VESSEL2::clbkPostStep (double *simt*, double *simdt*, double *mjd*) [virtual]**

Time step notification after state update.

Called at each simulation time step after the state has been updated to the current simulation time. This function allows to define actions which need to be controlled continuously.

**Parameters:**

*simt* current simulation run time [s]

*simdt* last time step length [s]

*mjd* absolute simulation time (days) in Modified Julian Date format.

**Default action:**

None.

**Note:**

This function, if implemented, is called at each frame for each instance of this vessel class, and is therefore time-critical. Avoid any unnecessary calculations here which may degrade performance.

**See also:**

[clbkPreStep](#)

**16.52.3.9 virtual bool VESSEL2::clbkPlaybackEvent (double *simt*, double *event\_t*, const char \* *event\_type*, const char \* *event*) [virtual]**

Playback event notification.

Called during playback of a recording session when a custom event tag in the vessel's articulation stream is encountered.

**Parameters:**

*simt* current simulation time [s]

*event\_t* recorded event time [s]

*event\_type* event tag string

*event* event data string

**Returns:**

Should return true if the event type is recognised and processed, false otherwise.

**Default action:**

Do nothing, return false.

**Note:**

This function can be used to process any custom vessel events that have been recorded with [VESSEL::RecordEvent](#) during a recording session.

**16.52.3.10 virtual void VESSEL2::clbkVisualCreated (VISHANDLE *vis*, int *refcount*) [virtual]**

Called after a vessel visual has been created by the renderer.

**Parameters:**

*vis* handle for the newly created visual  
*refcount* visual reference count

**Default action:**

None.

**Note:**

The logical interface to a vessel exists as long as the vessel is present in the simulation. However, the visual interface exists only when the vessel is within visual range of the camera. Orbiter creates and destroys visuals as required. This enhances simulation performance in the presence of a large number of objects in the simulation.

Whenever Orbiter creates a vessel's visual it reverts to its initial configuration (e.g. as defined in the mesh file). The module can use this function to update the visual to the current state, wherever dynamic changes are required.

More than one visual representation of an object may exist. The refcount parameter defines how many visual interfaces to the object exist.

**16.52.3.11 virtual void VESSEL2::clbkVisualDestroyed (VISHANDLE *vis*, int *refcount*) [virtual]**

Called before a vessel visual is destroyed.

**Parameters:**

*vis* handle for the visual to be destroyed  
*refcount* visual reference count

**Default action:**

None.

**Note:**

Orbiter calls this function before it destroys a visual representation of the vessel. This may be in response to the destruction of the actual vessel, but in general simply means that the vessel has moved out of visual range of the current camera location.

**16.52.3.12 virtual void VESSEL2::clbkDrawHUD (int *mode*, const HUDPAINTSPEC \**hps*, HDC *hDC*) [virtual]**

HUD redraw notification.

Called when the vessel's head-up display (HUD) needs to be redrawn (usually at each time step, unless the HUD is turned off). Overwriting this function allows to implement vessel-specific modifications of the HUD display (or to suppress the HUD altogether).

**Parameters:**

*mode* HUD mode (see `HUD_*` constants in [OrbiterAPI.h](#))

*hps* pointer to a `HUDPAINTSPEC` structure

*hDC* GDI drawing device context

**Default action:**

Draws a standard HUD display with Orbiter's default display layout.

**Deprecated**

This method contains a device-dependent drawing context and may not work with all graphics clients. It has been superseded by [VESSEL3::clbkDrawHUD](#).

**Note:**

For vessels derived from [VESSEL3](#) orbiter will not call this method, but will call the [VESSEL3::clbkDrawHUD](#) method instead. The [VESSEL3](#) version uses a generic *Sketchpad* drawing context instead of a HDC.

**See also:**

[VESSEL3::clbkDrawHUD](#)

**16.52.3.13 virtual void VESSEL2::clbkRCSMode (int mode) [virtual]**

Reaction Control System mode change notification.

Called when a vessel's RCS (reaction control system) mode changes. Usually the RCS consists of a set of small thrusters arranged so as to allow controlled attitude changes. In Orbiter, the RCS can be driven in either rotational mode (to change the vessel's angular velocity) or in linear mode (to change its linear velocity), or be switched off.

**Parameters:**

*mode* new RCS mode: 0=disabled, 1=rotational, 2=linear

**Default action:**

None.

**Note:**

This callback function is invoked when the user switches RCS mode via the keyboard ("/" or "Ctrl-/" on numerical keypad) or after a call to [VESSEL::SetAttitudeMode](#) or [VESSEL::ToggleAttitudeMode](#). Not all vessel types may support a reaction control system. In that case, the callback function can be ignored by the module.

**16.52.3.14 virtual void VESSEL2::clbkADCtrlMode (DWORD mode) [virtual]**

Aerodynamic control surface mode change notification.

Called when user input mode for aerodynamic control surfaces (elevator, rudder, aileron) changes.

**Parameters:**

*mode* control mode

**Default action:**

None.

**Note:**

The returned control mode contains bit flags as follows:

- bit 0: elevator enabled/disabled
- bit 1: rudder enabled/disabled
- bit 2: ailerons enabled/disabled

Therefore, mode=0 indicates control surfaces disabled, mode=7 indicates fully enabled.

**16.52.3.15 virtual void VESSEL2::clbkHUDMode (int *mode*) [virtual]**

HUD mode change notification.

Called after a change of the vessel's HUD (head-up-display) mode.

**Parameters:**

*mode* new HUD mode

**Default action:**

None.

**Note:**

For currently supported HUD modes see `HUD_*` constants in [OrbiterAPI.h](#)  
mode `HUD_NONE` indicates that the HUD has been turned off.

**See also:**

Section [HUD mode identifiers](#) for a list of default mode identifiers.

**16.52.3.16 virtual void VESSEL2::clbkMFDMode (int *mfd*, int *mode*) [virtual]**

MFD mode change modification.

Called when the user has switched one of the [MFD](#) (multi-functional display) instruments to a different display mode.

**Parameters:**

*mfd* MFD instrument identifier

*mode* new MFD mode identifier

**Default action:**

None.

**Note:**

This callback function can be used to refresh the [MFD](#) button labels after the [MFD](#) mode has changed, or if a mode requires a dynamic label update.

The mode parameter can be one of the [MFD](#) mode identifiers `MFD_*` listed in [OrbiterAPI.h](#), or `MFD_REFRESHBUTTONS`. The latter is sent as a result of a call to `oapiRefreshMFDButtons`. It indicates not a mode change, but the need to refresh the button labels within a mode (i.e. a mode that dynamically changed its labels).

**See also:**

Section [MFD mode identifiers](#) for a list of default mode identifiers.

**16.52.3.17 virtual void VESSEL2::clbkNavMode (int *mode*, bool *active*) [virtual]**

Navigation mode change notification.

Called when an automated "navigation mode" is activated or deactivated for a vessel. Most navigation modes engage the vessel's RCS to attain a specific attitude, including pro/retrograde, normal to the orbital plane, level with the local horizon, etc.

**Parameters:**

*mode* navmode identifier

*active* true if activated, false if deactivated

**Default action:**

None.

**See also:**

Section [Navigation mode identifiers](#) for a list of available navigation modes.

**16.52.3.18 virtual void VESSEL2::clbkDockEvent (int *dock*, OBJHANDLE *mate*) [virtual]**

Docking event notification.

Called after a docking or undocking event at one of the vessel's docking ports.

**Parameters:**

*dock* docking port index

*mate* handle to docked vessel, or NULL for undocking event

**Default action:**

None.

**Note:**

*dock* is the index ( $\geq 0$ ) of the vessel's docking port at which the docking/undocking event takes place.  
*mate* is a handle to the vessel docking at the port, or NULL to indicate an undocking event.

**16.52.3.19 virtual void VESSEL2::clbkAnimate (double *simt*) [virtual]**

Manual animation notification.

Called at each simulation time step if the module has registered at least one animation notification request and if the vessel's visual exists.

**Parameters:**

*simt* simulation time [s]

**Default action:**

None.

**Note:**

This callback allows the module to animate the vessel's visual representation (moving undercarriage, cargo bay doors, etc.)

It is only called as long as the vessel has registered an animation request (between matching [VESSEL::RegisterAnimation](#) and [VESSEL::UnregisterAnimation](#) calls) and if the vessel's visual exists.

This callback is *not* used for the "semi-automatic" animation mechanism ([VESSEL::CreateAnimation](#), [VESSEL::AddAnimationComponent](#))

**See also:**

[VESSEL::RegisterAnimation](#), [VESSEL::UnregisterAnimation](#), [VESSEL::CreateAnimation](#), [VESSEL::AddAnimationComponent](#)

**16.52.3.20 virtual int VESSEL2::clbkConsumeDirectKey (char \* kstate) [virtual]**

Keyboard status notification.

Called at each simulation time step to allow the module to query the current keyboard status. This callback can be used to install a custom keyboard interface for the vessel.

**Parameters:**

*kstate* keyboard state

**Returns:**

A nonzero return value will completely disable default processing of the key state for the current time step. To disable the default processing of selected keys only, use the RESETKEY macro (see [OrbiterAPI.h](#)) and return 0.

**Default action:**

None, returns 0.

**Note:**

The keystate contains the current keyboard state. Use the KEYDOWN macro in combination with the key identifiers as defined in [OrbiterAPI.h](#) (OAPI\_KEY\_\*) to check for particular keys being pressed.  
Example:

```
if (KEYDOWN (kstate, OAPI_KEY_F10)) {
    // perform action
    RESETKEY (kstate, OAPI_KEY_F10);
    // optional: prevent default processing of the key
}
```

This function should be used where a key state, rather than a key event is required, for example when engaging thrusters or similar. To test for key events (key pressed, key released) use [clbkConsumeBufferedKey\(\)](#) instead.

**16.52.3.21 virtual int VESSEL2::clbkConsumeBufferedKey (DWORD *key*, bool *down*, char \* *kstate*) [virtual]**

Keyboard event notification.

This callback function notifies the vessel of a buffered key event (key pressed or key released).

**Parameters:**

*key* key scan code (see OAPI\_KEY\_\* constants in [OrbiterAPI.h](#))

*down* true if key was pressed, false if key was released

*kstate* current keyboard state

**Returns:**

The function should return 1 if Orbiter's default processing of the key event should be skipped, 0 otherwise.

**Default action:**

None, returns 0.

**Note:**

The key state (*kstate*) can be used to test for key modifiers (Shift, Ctrl, etc.). The KEYMOD\_xxx macros defined in [OrbiterAPI.h](#) are useful for this purpose.

This function may be called repeatedly during a single frame, if multiple key events have occurred in the last time step.

**16.52.3.22 virtual bool VESSEL2::clbkLoadGenericCockpit () [virtual]**

Generic cockpit view mode request notification.

Called when the vessel's generic "glass cockpit" view (consisting of two "floating" **MFD** instruments and a HUD, displayed on top of the 3-D render window) is selected by the user pressing F8, or by a function call.

**Returns:**

The function should return true if it supports generic cockpit view, false otherwise.

**Default action:**

Sets camera direction to "forward" (0,0,1) and returns true.

**Note:**

The generic cockpit view is available for all vessel types by default, unless this function is overwritten to return false.

Only disable the generic view if the vessel supports either 2-D instrument panels (see [clbkLoadPanel](#)) or a virtual cockpit (see [clbkLoadVC](#)). If no valid cockpit view at all is available for a vessel, Orbiter will crash.

Even if the vessel supports panels or virtual cockpits, you shouldn't normally disable the generic view, because it provides the best performance on slower computers.

**See also:**

[clbkLoadPanel](#), [clbkLoadVC](#)

**16.52.3.23 virtual bool VESSEL2::clbkLoadPanel (int *id*) [virtual]**

2-D instrument panel view mode request notification

Called when Orbiter tries to switch the cockpit view to a 2-D instrument panel.

**Parameters:**

*id* panel identifier ( $\geq 0$ )

**Returns:**

The function should return true if it supports the requested panel, false otherwise.

**Default action:**

None, returns false.

**Note:**

In the body of this function the module should define the panel background bitmap and panel capabilities, e.g. the position of MFDs and other instruments, active areas (mouse hotspots) etc.

A vessel which implements panels must at least support panel id 0 (the main panel). If any panels register neighbour panels (see oapiSetPanelNeighbours), all the neighbours must be supported, too.

**See also:**

[oapiRegisterPanelBackground](#), [oapiRegisterPanelArea](#), [oapiRegisterMFD](#), [clbkLoadGenericCockpit](#), [clbkLoadVC](#)

**16.52.3.24 virtual bool VESSEL2::clbkPanelMouseEvent (int *id*, int *event*, int *mx*, int *my*) [virtual]**

Mouse event notification for 2-D panel views.

Called when a mouse-activated panel area receives a mouse event.

**Parameters:**

*id* panel area identifier

*event* mouse event (see [Mouse event identifiers](#))

*mx,my* relative mouse position in area at event

**Returns:**

The function should return true if it processes the event, false otherwise.

**Default action:**

None, returns false.

**Note:**

Mouse events are only sent for areas which requested notification during definition (see [oapiRegisterPanelArea](#)).

**16.52.3.25 virtual bool VESSEL2::clbkPanelRedrawEvent (int *id*, int *event*, SURFHANDLE *surf*) [virtual]**

Redraw event notification for 2-D panel views.

Called when a registered panel area needs to be redrawn.

**Parameters:**

*id* panel area identifier

*event* redraw event (see [Panel redraw event identifiers](#))

*surf* area surface handle

**Returns:**

The function should return true if it processes the event, false otherwise.

**Default action:**

None, returns false.

**Note:**

This callback function is only called for areas which were not registered with the PANEL\_REDRAW\_NEVER flag.

All redrawable panel areas receive a PANEL\_REDRAW\_INIT redraw notification when the panel is created, in addition to any registered redraw notification events.

The surface handle *surf* contains either the current area state, or the area background, depending on the flags passed during area registration.

The surface handle may be used for blitting operations, or to receive a Windows device context (DC) for Windows-style redrawing operations.

**See also:**

[oapiGetDC](#), [oapiReleaseDC](#), [oapiTriggerPanelRedrawArea](#)

**16.52.3.26 virtual bool VESSEL2::clbkLoadVC (int *id*) [virtual]**

3-D virtual cockpit view mode request notification

Called when Orbiter tries to switch the cockpit view to a 3-D virtual cockpit mode (for example in response to the user switching cockpit modes with F8).

**Parameters:**

*id* virtual cockpit identifier ( $\geq 0$ )

**Returns:**

true if the vessel supports the requested virtual cockpit, false otherwise.

**Default action:**

None, returning false (i.e. virtual cockpit mode not supported).

**Note:**

Multiple virtual cockpit camera positions (e.g. for pilot and co-pilot) can be defined. In this case, the body of clbkLoadVC should examine the value of *id* and set the VC parameters accordingly.

Multiple positions are defined by specifying the neighbour positions of the current position via a call to oapiVCSetNeighbours.

In the body of this function the module should define **MFD** display targets (with oapiVCRegisterMFD) and other active areas (with oapiVCRegisterArea) for the requested virtual cockpit.

**See also:**

[clbkLoadGenericCockpit](#), [clbkLoadPanel](#), [oapiVCSetNeighbours](#), [oapiVCRegisterArea](#)

### 16.52.3.27 **virtual bool VESSEL2::clbkVCMouseEvent (int *id*, int *event*, VECTOR3 & *p*) [virtual]**

Mouse event notification for 3-D virtual cockpit views.

Called when a mouse-activated virtual cockpit area receives a mouse event.

**Parameters:**

- id*** area identifier
- event*** mouse event (see [Mouse event identifiers](#))
- p*** parameter vector (area type-dependent, see notes)

**Returns:**

The function should return true if it processes the event, false otherwise.

**Default action:**

None, returning false.

**Note:**

To generate a mouse-activated area in a virtual cockpit, you must do the following when registering the area during clbkLoadVC:

- register the area with a call to oapiVCRegisterArea with a mouse mode other than PANEL\_MOUSE\_IGNORE.
- define a mouse-click area in the vessel's local frame. Use one of the oapiVCRegisterAreaClickmode\_XXX functions. You can define spherical or quadrilateral click areas.

Parameter p returns information about the mouse position at the mouse event. The type of information returned depends on the area type for which the event was generated:

- spherical area:
  - p.x is distance of mouse event from area centre
  - p.y and p.z not used
- quadrilateral area:
  - p.x and p.y are the area-relative mouse x and y positions (top left = (0,0), bottom right = (1,1))
  - p.z not used

**See also:**

[clbkLoadVC](#), [clbkPanelMouseEvent](#), [oapiVCRegisterArea](#)

**16.52.3.28 virtual bool VESSEL2::clbkVCRedrawEvent (int *id*, int *event*, SURFHANDLE *surf*)  
[virtual]**

Redraw event notification for 3-D virtual cockpit views.

Called when a registered virtual cockpit area needs to be redrawn.

**Parameters:**

- id* area identifier
- event* redraw event (see [Panel redraw event identifiers](#))
- surf* associated texture handle

**Returns:**

The function should return true if it processes the event, false otherwise.

**Default action:**

None, returning false.

**Note:**

To allow an area of the virtual cockpit to be redrawn dynamically, the area must be registered with oapiVCRegisterArea during clbkLoadVC, using a redraw mode other than PANEL\_REDRAW\_NEVER.

When registering the area with oapiVCRegisterArea, you must also provide a handle to the texture onto which the redrawn surface is mapped. This texture must be part of the virtual cockpit mesh, and it must be listed in the mesh file with the 'D' ("dynamic") flag (see [3DModel.pdf](#)).

"Redrawing" an area is not limited to dynamically updating textures. It may also involve mesh transforms (e.g. to animate levers and switches rendered in 3D).

The documentation for this class was generated from the following files:

- Orbitersdk/include/[VesselAPI.h](#)
- Orbitersdk/doxygen/API\_reference/examples.cpp

## 16.53 VESSEL3 Class Reference

```
#include <VesselAPI.h>
```

Inheritance diagram for VESSEL3:

Collaboration diagram for VESSEL3:

### 16.53.1 Detailed Description

Callback extensions to the [VESSEL](#) class.

The [VESSEL3](#) class extends [VESSEL2](#) with additional functionality. Developers should use this class for new projects. Existing vessel addons can make use of the new features by switching the base class from [VESSEL2](#) to [VESSEL3](#).

#### Public Member Functions

- [\*\*VESSEL3\*\* \(OBJHANDLE hVessel, int fmodel=1\)](#)  
*Creates a [VESSEL3](#) interface for a vessel object.*
- [\*\*int SetPanelBackground \(PANELHANDLE hPanel, SURFHANDLE \\*hSurf, DWORD nsurf, MESHHANDLE hMesh, DWORD width, DWORD height, DWORD baseline=0, DWORD scrollflag=0\)\*\*](#)  
*Set the background surface for a 2-D instrument panel.*
- [\*\*int SetPanelScaling \(PANELHANDLE hPanel, double defscale, double extscale\)\*\*](#)  
*Set scaling factors for 2-D instrument panel.*
- [\*\*int RegisterPanelMFDGeometry \(PANELHANDLE hPanel, int MFD\\_id, int nmesh, int ngroup\)\*\*](#)  
*Define an [MFD](#) display in the panel mesh.*
- [\*\*int RegisterPanelArea \(PANELHANDLE hPanel, int id, const RECT &pos, const RECT &texpos, int draw\\_event, int mouse\\_event, int bkmode\)\*\*](#)  
*Register an area of the panel to receive mouse and redraw events.*
- [\*\*int RegisterPanelArea \(PANELHANDLE hPanel, int id, const RECT &pos, int draw\\_event, int mouse\\_event, SURFHANDLE surf=NULL, void \\*context=NULL\)\*\*](#)  
*Register an area of the panel to receive mouse and redraw events.*
- [\*\*virtual bool clbkPanelMouseEvent \(int id, int event, int mx, int my, void \\*context\)\*\*](#)  
*Mouse event notification for 2-D panel views.*
- [\*\*virtual bool clbkPanelRedrawEvent \(int id, int event, SURFHANDLE surf, void \\*context\)\*\*](#)  
*Redraw event notification for 2-D panel views.*

- virtual int `clbkGeneric` (int msgid=0, int prm=0, void \*context=NULL)  
*Generic multi-purpose callback function.*
- virtual bool `clbkLoadPanel2D` (int id, `PANELHANDLE` hPanel, DWORD viewW, DWORD viewH)  
*Request for a 2D instrument panel definition in cockpit view.*
- virtual bool `clbkDrawHUD` (int mode, const `HUDPAINTSPEC` \*hps, `oapi::Sketchpad` \*skp)  
*HUD redraw notification.*
- virtual void `clbkRenderHUD` (int mode, const `HUDPAINTSPEC` \*hps, `SURFHANDLE` hDefault-Tex)  
*HUD render notification.*
- virtual void `clbkGetRadiationForce` (const `VECTOR3` &mflux, `VECTOR3` &F, `VECTOR3` &pos)  
*Returns force due to radiation pressure.*

### 16.53.2 Constructor & Destructor Documentation

#### 16.53.2.1 VESSEL3::VESSEL3 (`OBJHANDLE hVessel, int fmodel = 1`)

Creates a `VESSEL3` interface for a vessel object.

**See also:**

[VESSEL2](#)

### 16.53.3 Member Function Documentation

#### 16.53.3.1 int VESSEL3::SetPanelBackground (`PANELHANDLE hPanel, SURFHANDLE * hSurf, DWORD nsurf, MESHHANDLE hMesh, DWORD width, DWORD height, DWORD baseline = 0, DWORD scrollflag = 0`)

Set the background surface for a 2-D instrument panel.

**Parameters:**

`hPanel` panel handle  
`hSurf` array of surface handles  
`nsurf` number of surfaces  
`hMesh` mesh handle defining the billboard geometry  
`width` panel width [pixel]  
`height` panel height [pixel]  
`baseline` base line for edge attachment  
`scrollflag` panel attachment and scrolling bitflags

**Returns:**

Always returns 0.

**Note:**

This method should be applied in the body of [clbkLoadPanel2D](#).

The mesh defines the size and layout of the billboard mesh used for rendering the panel surface. Its vertex coordinates are interpreted as transformed, i.e. in terms of screen coordinates (pixels). The z-coordinate should be zero. Normals are ignored. Texture coordinates define which part of the surfaces are rendered.

The groups are rendered in the order they appear in the mesh. Later groups cover earlier ones. Therefore the groups should be arranged from backmost to frontmost elements.

In the simplest case, the mesh consists of a single rectangular area (4 nodes, 2 triangles) and a single surface, but can be more elaborate.

The texture indices of the mesh groups (TexIdx) are interpreted as indices into the hSurf list (zero-based).

This method increases the reference counters for the surfaces, so the caller should release them at some point.

The surfaces can contain an alpha channel to handle transparency.

**16.53.3.2 int VESSEL3::SetPanelScaling (PANELHANDLE *hPanel*, double *defscale*, double *extscale*)**

Set scaling factors for 2-D instrument panel.

**Parameters:**

*hPanel* panel handle

*defscale* default scale factor

*extscale* additional scale factor

**Returns:**

Always returns 0.

**Note:**

The scaling factors define the scaling between mesh coordinates and screen pixels.

*defscale* is the default factor, *extscale* is an additional scale which can be selected by the user via the mouse wheel.

Examples: scale=1: one mesh unit corresponds to one screen pixel, scale=viewW/panelW: panel fits screen width

**16.53.3.3 int VESSEL3::RegisterPanelMFDGeometry (PANELHANDLE *hPanel*, int *MFD\_id*, int *nmesh*, int *ngroup*)**

Define an [MFD](#) display in the panel mesh.

**Parameters:**

*hPanel* panel handle

*MFD\_id* [MFD](#) identifier ( $\geq 0$ )

*nmesh* panel mesh index ( $\geq 0$ )

*ngroup* mesh group index ( $\geq 0$ )

**Returns:**

Always returns 0.

**Note:**

This method reserves a mesh group for rendering the contents of an [MFD](#) display. The group should define a square area (typically consisting of 4 nodes and 2 triangles) with appropriate texture coordinates. When rendering the panel, the texture for this group is set to the current contents of the [MFD](#) display.

The order of mesh groups defines the rendering order. To render the [MFD](#) display on top of the panel, define it as the last group in the mesh. Alternatively, the [MFD](#) can be rendered first, if the panel texture contains a transparent area through which to view the [MFD](#).

#### 16.53.3.4 int VESSEL3::RegisterPanelArea (PANELHANDLE *hPanel*, int *id*, const RECT & *pos*, const RECT & *texpos*, int *draw\_event*, int *mouse\_event*, int *bkmode*)

Register an area of the panel to receive mouse and redraw events.

**Deprecated**

This method has been superseded by [VESSEL4::RegisterPanelArea](#).

**Parameters:**

*hPanel* panel handle

*id* area identifier

*pos* area boundary coordinates (mesh coordinates)

*texpos* area boundary (texture coordinates)

*draw\_event* event flags for redraw event triggers (see [Panel redraw event identifiers](#))

*mouse\_event* event flags for mouse event triggers (see [Mouse event identifiers](#))

*bkmode* flag for texture background provided to redraw callback function (see [Panel area texture mapping identifiers](#))

**Returns:**

Always returns 0.

**Note:**

This method activates a rectangular area of the panel for receiving mouse and redraw events.

*pos* specifies the borders of the area in 'logical' coordinates (0,0,width,height) as specified by [SetPanelBackground](#). Registered mouse events within this area will trigger a call to [VESSEL2::clbkPanelMouseEvent](#).

If the area needs to be able to update the panel texture, it should pass an appropriate redraw flag in *draw\_event*, and specify the texture coordinates of the redraw area in *texpos*.

If the panel contains multiple background textures, only the first texture can be redrawn with this function. To redraw other textures in the background texture array, use [VESSEL4::RegisterPanelArea](#) instead.

For backward compatibility, this method automatically adds the PANEL\_REDRAW\_GDI and PANEL\_REDRAW\_SKETCHPAD flags to *draw\_event*. If GDI and/or Sketchpad access to the area drawing surface is not required, using [VESSEL4::RegisterPanelArea](#) can improve graphics performance.

**See also:**

[VESSEL4::RegisterPanelArea](#)

**16.53.3.5 int VESSEL3::RegisterPanelArea (PANELHANDLE *hPanel*, int *id*, const RECT & *pos*, int *draw\_event*, int *mouse\_event*, SURFHANDLE *surf* = NULL, void \* *context* = NULL)**

Register an area of the panel to receive mouse and redraw events.

**Parameters:**

- hPanel*** panel handle
- id*** area identifier
- pos*** area boundary coordinates (mesh coordinates)
- draw\_event*** event flags for redraw event triggers (see [Panel redraw event identifiers](#))
- mouse\_event*** event flags for mouse event triggers (see [Mouse event identifiers](#))
- surf*** surface handle passed to the redraw callback function
- context*** user-defined data passed to the mouse and redraw callback functions

**Returns:**

Always returns 0.

**Note:**

This version passes the provided surface handle directly to the redraw callback, rather making a copy of the area. This is useful if the area either doesn't need to modify any surfaces, or blits parts of the same surface (e.g. a texture that contains both the panel background and various elements (switches, dials, etc.) to be copied on top.

Since the surface returned to the redraw function is not restricted to the registered area, it is the responsibility of the caller not to draw outside the area.

The area boundaries defined in *pos* are only used for generating mouse events. If the area does not process mouse events (PANEL\_MOUSE\_IGNORE), the *pos* parameter is ignored.

The PANEL\_REDRAW\_GDI and PANEL\_REDRAW\_SKETCHPAD flags can not be used in the *draw\_event* parameter. If GDI or Sketchpad access is required during redraw events, either the surface *surf* must have been created with the appropriate attributes, or [VESSEL4::RegisterPanelArea](#) should be used instead.

**See also:**

[VESSEL4::RegisterPanelArea](#), [oapiCreateSurfaceEx](#)

**16.53.3.6 virtual bool VESSEL3::clbkPanelMouseEvent (int *id*, int *event*, int *mx*, int *my*, void \* *context*) [virtual]**

Mouse event notification for 2-D panel views.

Called when a mouse-activated panel area receives a mouse event.

**Parameters:**

- id*** panel area identifier
- event*** mouse event (see [Mouse event identifiers](#))
- mx, my*** relative mouse position in area at event
- context*** user-supplied pointer to context data (defined in [RegisterPanelArea](#))

**Returns:**

The function should return true if it processes the event, false otherwise.

**Default action:**

None, returns false.

**Note:**

If a vessel class overloads this method, it should return true. On a *false* return, Orbiter will try [VESSEL2::clbkPanelMouseEvent](#) instead.

Mouse events are only sent for areas which requested notification during definition (see [RegisterPanelArea](#)).

**See also:**

[RegisterPanelArea](#)

**16.53.3.7 virtual bool VESSEL3::clbkPanelRedrawEvent (int *id*, int *event*, SURFHANDLE *surf*, void \* *context*) [virtual]**

Redraw event notification for 2-D panel views.

Called when a registered panel area needs to be redrawn.

**Parameters:**

*id* panel area identifier

*event* redraw event (see [Panel redraw event identifiers](#))

*surf* area surface handle

*context* user-supplied pointer to context data (defined in [RegisterPanelArea](#))

**Returns:**

The function should return true if it processes the event, false otherwise.

**Default action:**

None, returns false.

**Note:**

This callback function is only called for areas which were not registered with the PANEL\_REDRAW\_NEVER flag.

If a vessel class overloads this method, it should return true. On a *false* return, Orbiter will try [VESSEL2::clbkPanelRedrawEvent](#) instead.

All drawable panel areas receive a PANEL\_DRAW\_INIT redraw notification when the panel is created, in addition to any registered redraw notification events.

The surface handle *surf* contains either the current area state, or the area background, depending on the flags passed during area registration.

The surface handle may be used for blitting operations, or to receive a Windows device context (DC) for Windows-style redrawing operations.

**See also:**

[RegisterPanelArea](#), [oapiGetDC](#), [oapiReleaseDC](#), [oapiTriggerPanelRedrawArea](#)

**16.53.3.8 virtual int VESSEL3::clbkGeneric (int *msgid* = 0, int *prm* = 0, void \* *context* = NULL) [virtual]**

Generic multi-purpose callback function.

**Parameters:**

*msgid* message identifier (see [Generic vessel message identifiers](#))

*prm* message parameter

*context* pointer to additional message data

**Returns:**

Result flag.

**16.53.3.9 virtual bool VESSEL3::clbkLoadPanel2D (int *id*, PANELHANDLE *hPanel*, DWORD *viewW*, DWORD *viewH*) [virtual]**

Request for a 2D instrument panel definition in cockpit view.

**Parameters:**

*id* panel identifier ( $\geq 0$ )

*hPanel* panel handle

*viewW* viewport width [pixel]

*viewH* viewport height [pixel]

**Returns:**

The function should return *true* if it supports the requested panel, false otherwise.

**Default action:**

None, returns false.

**Note:**

This method replaces [VESSEL2::clbkLoadPanel](#). It defines the panels via SURFHANDLES instead of bitmaps.

**16.53.3.10 virtual bool VESSEL3::clbkDrawHUD (int *mode*, const HUDPAINTSPEC \* *hps*, oapi::Sketchpad \* *skp*) [virtual]**

HUD redraw notification.

Called when the vessel's head-up display (HUD) needs to be redrawn (usually at each time step, unless the HUD is turned off). Overwriting this function allows to implement vessel-specific modifications of the HUD display (or to suppress the HUD altogether).

**Parameters:**

*mode* HUD mode (see *HUD\_\** constants in [OrbiterAPI.h](#))

*hps* pointer to a *HUDPAINTSPEC* structure (see notes)

*skp* drawing context instance

**Returns:**

Overloaded methods should return *true*. If the return value is *false*, orbiter assumes that this method is disabled and will try [VESSEL2::clbkDrawHUD](#).

**Default action:**

Draws a standard HUD display with Orbiter's default display layout and returns *true*.

**Note:**

If a vessel overwrites this method, Orbiter will draw the default HUD only if the base class [VESSEL3::clbkDrawHUD](#) is called.

*hps* points to a **HUDPAINTSPEC** structure containing information about the HUD drawing surface. It has the following format:

```
typedef struct {
    int W, H;
    int CX, CY;
    double Scale;
    int Markersize;
} HUDPAINTSPEC;
```

where *W* and *H* are width and height of the HUD drawing surface in pixels, *CX* and *CY* are the *x* and *y* coordinates of the HUD centre (the position of the "forward marker", which is not guaranteed to be in the middle of the drawing surface or even within the drawing surface!), *Scale* represents an angular aperture of 1 deg. expressed in HUD pixels, and *Markersize* is a "typical" size which can be used to scale objects like direction markers.

The device context passed to *clbkDrawHUD* contains the appropriate settings for the current HUD display (font, pen, colours). If you need to change any of the GDI settings, make sure to restore the defaults before calling the base class *clbkDrawHUD*. Otherwise the default display will be corrupted. *clbkDrawHUD* can be used to implement entirely new vessel-specific HUD modes. In this case, the module would maintain its own record of the current HUD mode, and ignore the mode parameter passed to *clbkDrawHUD*.

In glass cockpit and 2-D panel mode, the HUD display can be a combination of drawn elements (via *clbkDrawHUD*) and rendered elements (via [clbkRenderHUD](#)). In VC mode, the HUD is always drawn. To disable all default HUD display elements, a derived vessel should overload both *clbkDrawHUD* and *clbkRenderHUD*.

**See also:**

[clbkRenderHUD](#), [Section HUD mode identifiers](#) for a list of default mode identifiers.

### 16.53.3.11 virtual void VESSEL3::clbkRenderHUD (int *mode*, const **HUDPAINTSPEC** \* *hps*, **SURFHANDLE** *hDefaultTex*) [virtual]

HUD render notification.

Called when the vessel's head-up display (HUD) needs to be rendered (usually at each time step, unless the HUD is turned off). Overwriting this function allows to implement vessel-specific modifications of the HUD display (or to suppress the HUD altogether).

**Parameters:**

*mode* HUD mode (see **HUD\_\*** constants in [OrbiterAPI.h](#))

*hps* pointer to a **HUDPAINTSPEC** structure

*hDefaultTex* handle for default HUD texture

**Default action:**

Renders a standard HUD display with Orbiter's default display layout.

**Note:**

This function is only called in glass cockpit or 2-D panel mode, not in VC (virtual cockpit mode). In glass cockpit or 2-D panel mode, the programmer has a choice of using `clbkRenderHUD` or `clbkDrawHUD` to display vessel-specific HUD elements. The use of `clbkRenderHUD` is preferred, because it provides smoother animation, better performance and is better supported by external render engines.

To disable all default HUD display, a derived vessel class should overload both `clbkRenderHUD` and `clbkDrawHUD`.

To render custom HUD elements, the `oapiRenderHUD` function should be called from within this callback function.

**See also:**

`clbkDrawHUD`, `oapiRenderHUD`, Section [HUD mode identifiers](#) for a list of default mode identifiers.

**16.53.3.12 virtual void VESSEL3::clbkGetRadiationForce (const VECTOR3 & *mflux*, VECTOR3 & *F*, VECTOR3 & *pos*) [virtual]**

Returns force due to radiation pressure.

**Parameters:**

← *mflux* momentum flux vector [ $\text{N}/\text{m}^2$ ] at current spacecraft position, transformed into vessel frame  
 → *F* radiation force vector [ $\text{N}$ ] in vessel frame  
 → *pos* force attack point [ $\text{m}$ ] in vessel frame

**Default action:**

Sets  $F = mflux * \text{size}^2 * a$ , where  $a$  (albedo coefficient) is fixed to 1.5. Sets  $pos = (0,0,0)$ . This simple formula ignores any attitude-dependent variations in surface area, and any non-radial force components due to oblique reflections. Does not induce any torque. For more sophisticated treatment, vessels should re-implement this method.

**Note:**

This method is called by orbiter when perturbation forces due to radiation pressure need to be evaluated. The implementation should take into account geometric factors (cross sections), surface factors (absorption, reflection) and spacecraft attitude relative to the sun.

The momentum flux parameter, *mflux*, takes into account shadow effects from the closest planet, or from the closest moon and its parent planet, if applicable.

If the returned force attack point *pos* is not set to the centre of gravity, (0,0,0), then a torque may be induced as well as a linear force.

If the vessel contains multiple distinct surfaces, the returned force should be the vector sum of all individual contributions, and the returned position should be the weighted barycentre of all individual contributions w.r.t. the vessel centre of gravity.

The documentation for this class was generated from the following file:

- OrbiterSDK/include/[VesselAPI.h](#)

## 16.54 VESSEL4 Class Reference

```
#include <VesselAPI.h>
```

Inheritance diagram for VESSEL4:Collaboration diagram for VESSEL4:

### 16.54.1 Detailed Description

Extensions to the [VESSEL](#) class.

The [VESSEL4](#) class extends [VESSEL3](#) with additional functionality. Developers should use this class for new projects. Existing vessel addons can make use of the new features by switching the base class to [VESSEL4](#).

### Public Member Functions

- [VESSEL4 \(OBJHANDLE hVessel, int fmodel=1\)](#)  
*Creates a [VESSEL4](#) interface for a vessel object.*
- [int RegisterPanelArea \(PANELHANDLE hPanel, int id, const RECT &pos, int texidx, const RECT &texpos, int draw\\_event, int mouse\\_event, int bkmode\)](#)  
*Register an area of the panel to receive mouse and redraw events.*

### 16.54.2 Constructor & Destructor Documentation

#### 16.54.2.1 VESSEL4::VESSEL4 (OBJHANDLE *hVessel*, int *fmodel* = 1)

Creates a [VESSEL4](#) interface for a vessel object.

See also:

[VESSEL3](#)

### 16.54.3 Member Function Documentation

#### 16.54.3.1 int VESSEL4::RegisterPanelArea (PANELHANDLE *hPanel*, int *id*, const RECT & *pos*, int *texidx*, const RECT & *texpos*, int *draw\_event*, int *mouse\_event*, int *bkmode*)

Register an area of the panel to receive mouse and redraw events.

#### Parameters:

- hPanel* panel handle
- id* area identifier
- pos* area boundary coordinates (mesh coordinates)
- texidx* background texture index
- texpos* area boundary (texture coordinates)
- draw\_event* event flags for redraw event triggers (see [Panel redraw event identifiers](#))
- mouse\_event* event flags for mouse event triggers (see [Mouse event identifiers](#))
- bkmode* flag for texture background provided to redraw callback function (see [Panel area texture mapping identifiers](#))

**Returns:**

Always returns 0.

**Note:**

This method activates a rectangular area of the panel for receiving mouse and redraw events. *pos* specifies the borders of the area in 'logical' coordinates (0,0,width,height) as specified by [SetPanelBackground](#). Registered mouse events within this area will trigger a call to [VESSEL2::clbkPanelMouseEvent](#).

*texidx* is the index of the panel background texture the area texture should be copied into, in the order the textures were specified in the array passed to [VESSEL3::SetPanelBackground](#). If only a single texture is used for the panel, *texidx* should be set to 0. If the area doesn't need to be redrawn (PANEL\_REDRAW\_NEVER), this parameter is ignored.

If the area texture should allow GDI and/or Sketchpad access during redraw events, the PANEL\_REDRAW\_GDI and/or PANEL\_REDRAW\_SKETCHPAD flags should be added to *draw\_event*. If only blitting access is required, these flags should be omitted for improved performance.

The documentation for this class was generated from the following file:

- OrbiterSDK/include/[VesselAPI.h](#)

## 16.55 VESSELSTATUS Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for VESSELSTATUS:

### 16.55.1 Detailed Description

Vessel status parameters (version 1).

Defines vessel status parameters at a given time. This is version 1 of the vessel status interface. It is retained for backward compatibility, but new modules should use [VESSELSTATUS2](#) instead to exploit the latest vessel capabilities such as individual thruster and propellant resource settings.

#### Public Attributes

- [VECTOR3 rpos](#)  
*position relative to rbody in ecliptic frame [m]*
- [VECTOR3 rvel](#)  
*velocity relative to rbody in ecliptic frame [m/s]*

- **VECTOR3 vrot**  
*rotation velocity about principal axes in ecliptic frame [rad/s]*
- **VECTOR3 arot**  
*vessel orientation against ecliptic frame*
- double **fuel**  
*fuel level [0..1]*
- double **eng\_main**  
*main/retro engine setting [-1..1]*
- double **eng\_hovr**  
*hover engine setting [0..1]*
- **OBJHANDLE rbody**  
*handle of reference body*
- **OBJHANDLE base**  
*handle of docking or landing target*
- int **port**  
*index of designated docking or landing port*
- int **status**  
*flight status indicator*
- **VECTOR3 vdata [10]**  
*additional vector parameters*
- double **fdata [10]**  
*additional floating point parameters (not used)*
- DWORD **flag [10]**  
*additional integer and bitflag parameters*

## 16.55.2 Member Data Documentation

### 16.55.2.1 int VESSELSTATUS::status

flight status indicator

**Note:**

- 0=active (freeflight)
- 1=inactive (landed)

**16.55.2.2 VECTOR3 VESSELSTATUS::vdata[10]**

additional vector parameters

**Note:**

- vdata[0]: contains landing paramters if status == 1: vdata[0].x = longitude, vdata[0].y = latitude, vdata[0].z = heading of landed vessel
- vdata[1] - vdata[9]: not used

**16.55.2.3 DWORD VESSELSTATUS::flag[10]**

additional integer and bitflag parameters

**flag[0]&1:**

- 0: ingore eng\_main and eng\_hovr entries, do not change thruster settings
- 1: set THGROUP\_MAIN and THGROUP\_RETRO thruster groups from eng\_main, and THGROUP\_HOVER from eng\_hovr.

**flag[0]&2:**

- 0: ignore fuel level, do not change fuel levels
- 1: set fuel level of first propellant resource from fuel

**Note:**

flag[1] - flag[9]: not used

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

**16.56 VESSELSTATUS2 Struct Reference**

```
#include <OrbiterAPI.h>
```

Collaboration diagram for VESSELSTATUS2:

### 16.56.1 Detailed Description

Vessel status parameters (version 2).

Defines vessel status parameters at a given time. This is version 2 of the vessel status interface and replaces the earlier [VESSELSTATUS](#) structure. Functions using [VESSELSTATUS](#) are still supported for backward compatibility.

**Note:**

The version specification is an input parameter for all function calls (including GetStatus) and must be set by the user to tell Orbiter which interface to use.

**See also:**

[VESSEL::GetStatusEx](#)

#### Public Attributes

- **DWORD version**  
*interface version identifier (2)*
- **DWORD flag**  
*bit flags*
- **OBJHANDLE rbody**  
*handle of reference body*
- **OBJHANDLE base**  
*handle of docking or landing target*
- **int port**  
*index of designated docking or landing port*
- **int status**  
*flight status indicator*
- **VECTOR3 rpos**  
*position relative to reference body (rbody) in ecliptic frame [m]*
- **VECTOR3 rvel**  
*velocity relative to reference body in ecliptic frame [m/s]*
- **VECTOR3 vrot**  
*angular velocity around principal axes in ecliptic frame [rad/s]*
- **VECTOR3 arot**  
*vessel orientation against ecliptic frame*
- **double surf\_lng**  
*longitude of vessel position in equatorial coordinates of rbody [rad]*

- double `surf_lat`  
*latitude of vessel position in equatorial coordinates of rbody [rad]*
- double `surf_hdg`  
*vessel heading on the ground [rad]*
- DWORD `nfuel`  
*number of entries in the fuel list*
- struct `VESSELSTATUS2::FUELSPEC * fuel`  
*propellant list*
- DWORD `nthruster`  
*number of entries in the thruster list*
- struct `VESSELSTATUS2::THRUSTSPEC * thruster`  
*thruster definition list*
- DWORD `ndockinfo`  
*number of entries in the dockinfo list*
- struct `VESSELSTATUS2::DOCKINFOSPEC * dockinfo`  
*dock info list*
- DWORD `xpdr`  
*transponder channel [0...640]*

## Classes

- struct `DOCKINFOSPEC`  
*dock info list*
- struct `FUELSPEC`  
*propellant list*
- struct `THRUSTSPEC`  
*thruster definition list*

## 16.56.2 Member Data Documentation

### 16.56.2.1 DWORD VESSELSTATUS2::flag

bit flags

The meaning of the bitflags in flag depends on whether the `VESSELSTATUS2` structure is used to get (GetStatus) or set (SetStatus) a vessel status. The following flags are currently defined:

**flags:**

- `VS_FUELRESET`

- Get - not used
- Set - reset all fuel levels to zero, independent of the fuel list.
- VS\_FUELLIST
  - Get - request a list of current fuel levels in fuel. The module is responsible for deleting the list after use.
  - Set - set fuel levels for all resources listed in fuel.
- VS\_THRUSTRESET
  - Get - not used
  - Set - reset all thruster levels to zero, independent of the thruster list
- VS\_THRUSTLIST
  - Get - request a list of current thrust levels in thruster. The module is responsible for deleting the list after use.
  - Set - set thrust levels for all thrusters listed in thruster.
- VS.DockInfoList
  - Get - request a docking port status list in dockinfo. The module is responsible for deleting the list after use.
  - Set - initialise docking status for all docking ports in dockinfo.

**See also:**

[VESSEL::GetStatusEx](#)

**16.56.2.2 int VESSELSTATUS2::status**

flight status indicator

**Note:**

- 0=active (freeflight)
- 1=inactive (landed)

**16.56.2.3 VECTOR3 VESSELSTATUS2::arot**

vessel orientation against ecliptic frame

**arot** ( $\alpha, \beta, \gamma$ ) contains angles of rotation [rad] around  $x, y, z$  axes in ecliptic frame to produce this rotation matrix **R** for mapping from the vessel's local frame of reference to the global frame of reference:

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & \sin \alpha \\ 0 & -\sin \alpha & \cos \alpha \end{bmatrix} \begin{bmatrix} \cos \beta & 0 & -\sin \beta \\ 0 & 1 & 0 \\ \sin \beta & 0 & \cos \beta \end{bmatrix} \begin{bmatrix} \cos \gamma & \sin \gamma & 0 \\ -\sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

such that  $\mathbf{r}_{\text{global}} = \mathbf{R} \mathbf{r}_{\text{local}} + \mathbf{p}$

where **p** is the vessel's global position.

**16.56.2.4 double VESSELSTATUS2::surf\_lng**

longitude of vessel position in equatorial coordinates of rbody [rad]

**Note:**

currently only defined if the vessel is landed (status=1)

**16.56.2.5 double VESSELSTATUS2::surf\_lat**

latitude of vessel position in equatorial coordinates of rbody [rad]

**Note:**

currently only defined if the vessel is landed (status=1)

**16.56.2.6 double VESSELSTATUS2::surf\_hdg**

vessel heading on the ground [rad]

**Note:**

currently only defined if the vessel is landed (status=1)

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

**16.57 VESSELSTATUS2::DOCKINFOSPEC Struct Reference**

```
#include <OrbiterAPI.h>
```

**16.57.1 Detailed Description**

dock info list

**Public Attributes**

- DWORD **idx**  
*docking port index*
- DWORD **ridx**  
*docking port index of docked vessel*
- [OBJHANDLE rvessel](#)  
*docked vessel*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

**16.58 VESSELSTATUS2::FUELSPEC Struct Reference**

```
#include <OrbiterAPI.h>
```

### 16.58.1 Detailed Description

propellant list

#### Public Attributes

- DWORD **idx**  
*propellant index*
- double **level**  
*propellant level*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 16.59 VESSELSTATUS2::THRUSTSPEC Struct Reference

```
#include <OrbiterAPI.h>
```

### 16.59.1 Detailed Description

thruster definition list

#### Public Attributes

- DWORD **idx**  
*thruster index*
- double **level**  
*thruster level*

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

## 17 File Documentation

### 17.1 Orbitersdk/include/CelBodyAPI.h File Reference

#### 17.1.1 Detailed Description

Contains interface classes for celestial bodies: [CELBODY](#) and [CELBODY2](#).

## Classes

- class [CELBODY](#)  
*This is the base class for celestial body classes.*
- class [CELBODY2](#)  
*Extension to [CELBODY](#) class.*
- class [ATMOSPHERE](#)  
*Defines the physical atmospheric properties for a celestial body.*
- struct [ATMOSPHERE::PRM\\_IN](#)  
*Input parameters for atmospheric data calculation.*
- struct [ATMOSPHERE::PRM\\_OUT](#)  
*Output parameters for atmospheric data calculation.*

## Defines

- #define [Ephem\\_TruePos](#) 0x01  
*true body position*
- #define [Ephem\\_TrueVel](#) 0x02  
*true body velocity*
- #define [Ephem\\_BaryPos](#) 0x04  
*barycentric position*
- #define [Ephem\\_BaryVel](#) 0x08  
*barycentric velocity*
- #define [Ephem\\_BaryIsTrue](#) 0x10  
*body has no child objects*
- #define [Ephem\\_ParentBary](#) 0x20  
*ephemerides are computed in terms of the barycentre of the parent body's system*
- #define [Ephem\\_Polar](#) 0x40  
*data is returned in polar format*

## 17.2 Orbitersdk/include/DrawAPI.h File Reference

### 17.2.1 Detailed Description

2-D surface drawing support interface.

```
#include "OrbiterAPI.h"
```

Include dependency graph for DrawAPI.h:

## Namespaces

- namespace **oapi**

## Classes

- union **oapi::IVECTOR2**  
*Integer-valued 2-D vector type.*
- class **oapi::DrawingTool**  
*Base class for various 2-D drawing resources (fonts, pens, brushes, etc.).*
- class **oapi::Font**  
*A font resource for drawing text. A font has a defined size, typeface, slant, weight, etc. Fonts can be selected into a [Sketchpad](#) and then apply to all subsequent Text calls.*
- class **oapi::Pen**  
*A pen is a resource used for drawing lines and the outlines of closed figures such as rectangles, ellipses and polygons.*
- class **oapi::Brush**  
*A brush is a drawing resource for filling closed figures (rectangles, ellipses, polygons).*
- class **oapi::Sketchpad**  
*A Sketchpad object defines an environment for drawing onto 2-D surfaces.*

## 17.3 Orbitersdk/include/MFDAPI.h File Reference

### 17.3.1 Detailed Description

Class interfaces for [MFD](#) instruments and [MFD](#) modes.

```
#include "OrbiterAPI.h"
```

Include dependency graph for MFDAPIL.h:

## Classes

- class [MFD](#)

*This class acts as an interface for user defined MFD (multi functional display) modes.*

- class [MFD2](#)

*Extended [MFD](#) class.*

- class [GraphMFD](#)

*This class is derived from [MFD](#) and provides a template for [MFD](#) modes containing 2D graphs.*

- class [ExternMFD](#)

*[ExternMFD](#) provides support for defining an [MFD](#) display in a plugin module.*

## 17.4 Orbitersdk/include/OrbiterAPI.h File Reference

### 17.4.1 Detailed Description

General API interface functions.

#### [Todo](#)

Check functions in [VESSELSTATUS2::arot](#) and [oapiGetPlanetObliquityMatrix\(\)](#), minus sign has changed a place in a matrix. Is this correct??

#### [Todo](#)

class CameraMode documentation

```
#include <fstream>
#include <windows.h>
#include <float.h>
#include <math.h>
#include "lua\lua.h"
```

Include dependency graph for OrbiterAPI.h:

This graph shows which files directly or indirectly include this file:

## Namespaces

- namespace **oapi**

## Classes

- union **VECTOR3**  
*3-element vector*
- union **MATRIX3**  
*3x3-element matrix*
- struct **COLOUR4**  
*colour definition*
- struct **NTVERTEX**  
*vertex definition including normals and texture coordinates*
- struct **MESHGROUP**  
*Defines a mesh group (subset of a mesh).*
- struct **MESHGROUPEX**  
*extended mesh group definition*
- struct **GROUPEDITSPEC**  
*Structure used by [oapiEditMeshGroup](#) to define the group elements to be replaced.*
- struct **MATERIAL**  
*material definition*
- struct **ELEMENTS**  
*Kepler orbital elements.*
- struct **ORBITPARAM**

*Secondary orbital parameters derived from the primary [ELEMENTS](#).*

- struct [ATMCONST](#)  
*Planetary atmospheric constants structure.*
- struct [ATMPARAM](#)  
*Atmospheric parameters structure.*
- struct [ENGINESTATUS](#)  
*Engine status.*
- struct [EXHAUSTSPEC](#)  
*Engine exhaust render parameters.*
- struct [PARTICLESTREAMSPEC](#)  
*Particle stream parameters.*
- class [LightEmitter](#)  
*Base class for defining a light source that can illuminate other objects.*
- class [PointLight](#)  
*Class for isotropic point light source.*
- class [SpotLight](#)  
*Class for directed spot light sources.*
- struct [NAVDATA](#)  
*Navigation transmitter data.*
- struct [BEACONLIGHTSPEC](#)  
*vessel beacon light parameters*
- struct [VESSELSTATUS](#)  
*Vessel status parameters (version 1).*
- struct [VESSELSTATUS2](#)  
*Vessel status parameters (version 2).*
- struct [VESSELSTATUS2::FUELSPEC](#)  
*propellant list*
- struct [VESSELSTATUS2::THRUSTSPEC](#)  
*thruster definition list*
- struct [VESSELSTATUS2::DOCKINFOSPEC](#)  
*dock info list*
- struct [LISTENTRY](#)  
*Entry specification for selection list entry.*

- struct **HELPCONTEXT**

*Context information for an Orbiter ingame help page.*

- struct **MESHGROUP\_TRANSFORM**

*This structure defines an affine mesh group transform (translation, rotation or scaling).*

- struct **ANIMATIONCOMP**

*Animation component definition.*

- struct **ANIMATION**

*Animation definition.*

- union **HUDPARAM**

*Mode-specific parameters for HUD mode settings.*

- class **LaunchpadItem**

*Base class to define launchpad items.*

## Defines

- #define **DLLEXPORT** \_\_declspec(dllexport)
- #define **DLLIMPORT** \_\_declspec(dllimport)
- #define **DLLCLBK** extern "C" \_\_declspec(dllexport)
- #define **OAPIFUNC** DLLIMPORT
- #define **RENDERTGT\_NONE** ((SURFHANDLE)-1)  
*no surface*

- #define **RENDERTGT\_MAINWINDOW** 0
- #define **OAPISURFACE\_TEXTURE** 0x0001

*Surface can be used as a texture (e.g. by associating it with a mesh).*

- #define **OAPISURFACE\_RENDERTARGET** 0x0002

*Surface can be rendered to by the graphics device.*

- #define **OAPISURFACE\_GDI** 0x0004

*A HDC context can be requested from the surface for GDI drawing.*

- #define **OAPISURFACE\_SKETCHPAD** 0x0008

*A Sketchpad context can be requested from the surface for Sketchpad drawing.*

- #define **OAPISURFACE\_MIPMAPS** 0x0010

*Create a full chain of mipmaps for the surface. If loaded from file, add any missing mipmap levels.*

- #define **OAPISURFACE\_NOMIPMAPS** 0x0020

*Don't create mipmaps. If loaded from file, ignore any mipmap levels present.*

- #define **OAPISURFACE\_ALPHA** 0x0040

*Create an alpha channel for the surface. If loaded from file, add an alpha channel if required.*

- #define **OAPISURFACE\_NOALPHA** 0x0080  
*Don't create an alpha channel. If loaded from file, strip any existing alpha channel.*
- #define **OAPISURFACE\_UNCOMPRESS** 0x0100  
*Create an uncompressed surface. If loaded from file, uncompress if required.*
- #define **OAPISURFACE\_SYSMEM** 0x0200  
*Create the surface in system (host) memory.*
- #define **GRPEDIT\_SETUSERFLAG** 0x0001  
*replace the group's UsrFlag entry with the value in the **GROUPEDITSPEC** structure.*
- #define **GRPEDIT\_ADDUSERFLAG** 0x0002  
*Add the UsrFlag value to the group's UsrFlag entry.*
- #define **GRPEDIT\_DELUSERFLAG** 0x0004  
*Remove the UsrFlag value from the group's UsrFlag entry.*
- #define **GRPEDIT\_VTXCRDX** 0x0008  
*Replace vertex x-coordinates.*
- #define **GRPEDIT\_VTXCRDY** 0x0010  
*Replace vertex y-coordinates.*
- #define **GRPEDIT\_VTXCRDZ** 0x0020  
*Replace vertex z-coordinates.*
- #define **GRPEDIT\_VTXCRD** (**GRPEDIT\_VTXCRDX** | **GRPEDIT\_VTXCRDY** | **GRPEDIT\_VTXCRDZ**)  
*Replace vertex coordinates.*
- #define **GRPEDIT\_VTXNMLX** 0x0040  
*Replace vertex x-normals.*
- #define **GRPEDIT\_VTXNMLY** 0x0080  
*Replace vertex y-normals.*
- #define **GRPEDIT\_VTXNMLZ** 0x0100  
*Replace vertex z-normals.*
- #define **GRPEDIT\_VTXNML** (**GRPEDIT\_VTXNMLX** | **GRPEDIT\_VTXNMLY** | **GRPEDIT\_VTXNMLZ**)  
*Replace vertex normals.*
- #define **GRPEDIT\_VTXTEXU** 0x0200  
*Replace vertex u-texture coordinates.*
- #define **GRPEDIT\_VTXTEXV** 0x0400  
*Replace vertex v-texture coordinates.*

- #define **GRPEDIT\_VTXTEX** (GRPEDIT\_VTXTEXU | GRPEDIT\_VTXTEXV)  
*Replace vertex texture coordinates.*
- #define **GRPEDIT\_VTX** (GRPEDIT\_VTXCRD | GRPEDIT\_VTXNML | GRPEDIT\_VTXTEX)  
*Replace vertices.*
- #define **EXHAUST\_CONSTANTLEVEL** 0x0001  
*exhaust level is constant*
- #define **EXHAUST\_CONSTANTPOS** 0x0002  
*exhaust position is constant*
- #define **EXHAUST\_CONSTANTDIR** 0x0004  
*exhaust direction is constant*
- #define **BEACONSHAPE\_COMPACT** 0  
*compact beacon shape*
- #define **BEACONSHAPE\_DIFFUSE** 1  
*diffuse beacon shape*
- #define **BEACONSHAPE\_STAR** 2  
*star-shaped beacon*
- #define **VS\_FUELRESET** 0x00000001  
*set all propellant levels to zero*
- #define **VS\_FUELLIST** 0x00000002  
*list of propellant levels is provided*
- #define **VS\_THRUSTRESET** 0x00000004  
*set all thruster levels to zero*
- #define **VS\_THRUSTLIST** 0x00000008  
*list of thruster levels is provided*
- #define **VS\_DOCKINFOLIST** 0x00000010  
*list of docked objects is provided*
- #define **LISTENTRY\_SUBITEM** 0x01  
*list entry has subitems*
- #define **LISTENTRY\_INACTIVE** 0x02  
*list entry can not be selected*
- #define **LISTENTRY\_SEPARATOR** 0x04  
*entry is followed by a separator*
- #define **LIST\_UPENTRY** 0x01  
*list has parent list*

- #define **LISTCLBK\_CANCEL** 0x00  
*user cancelled the selection list*
- #define **LISTCLBK\_SELECT** 0x01  
*user selected an item*
- #define **LISTCLBK\_SUBITEM** 0x02  
*user steps down to subitem*
- #define **LISTCLBK\_UPLIST** 0x03  
*user steps up to parent list*
- #define **LOCALVERTEXLIST** ((UINT)(-1))  
*flags animation component as explicit vertex list*
- #define **MAKEGROUPARRAY**(x) ((UINT\*)x)  
*casts a vertex array into a group*
- #define **MFD\_SHOWMODELABELS** 1
- #define **AIRCTRL\_AXIS\_AUTO** 0  
*Constants to define the rotation axis and direction of aerodynamic control surfaces.*
- #define **AIRCTRL\_AXIS\_YPOS** 1  
*y-axis (vertical), positive rotation*
- #define **AIRCTRL\_AXIS\_YNEG** 2  
*y-axis (vertical), negative rotation*
- #define **AIRCTRL\_AXIS\_XPOS** 3  
*x-axis (transversal), positive rotation*
- #define **AIRCTRL\_AXIS\_XNEG** 4  
*x-axis (transversal), negative rotation*
- #define **OBJTP\_INVALID** 0
- #define **OBJTP\_GENERIC** 1
- #define **OBJTP\_CBODY** 2
- #define **OBJTP\_STAR** 3
- #define **OBJTP\_PLANET** 4
- #define **OBJTP\_VESSEL** 10
- #define **OBJTP\_SURFBASE** 20
- #define **EVENT\_VESSEL\_INSMESH** 0  
*Insert a mesh (context: mesh index).*
- #define **EVENT\_VESSEL\_DELMESH** 1  
*Delete a mesh (context: mesh index, or -1 for all).*
- #define **EVENT\_VESSEL\_MESHVISMODE** 2  
*Set mesh visibility mode (context: mesh index).*

- #define **EVENT\_VESSEL\_RESETANIM** 3  
*Reset animations.*
- #define **EVENT\_VESSEL\_CLEARANIM** 4  
*Clear all animations (context: `UINT` (1=reset animations, 0=leave animations at current state)).*
- #define **EVENT\_VESSEL\_DELANIM** 5  
*Delete an animation (context: animation index).*
- #define **EVENT\_VESSEL\_NEWANIM** 6  
*Create a new animation (context: animation index).*
- #define **EVENT\_VESSEL\_MESHOFs** 7  
*Shift a mesh (context: mesh index).*
- #define **EVENT\_VESSEL\_MODMESHGROUP** 8  
*A mesh group has been modified.*
- #define **NAVMODE\_KILLROT** 1  
*"Kill rotation" mode*
- #define **NAVMODE\_HLEVEL** 2  
*"Hold level with horizon" mode*
- #define **NAVMODE\_PROGRADE** 3  
*"Prograde" mode*
- #define **NAVMODE\_RETROGRADE** 4  
*"Retrograde" mode*
- #define **NAVMODE\_NORMAL** 5  
*"Normal to orbital plane" mode*
- #define **NAVMODE\_ANTINORMAL** 6  
*"Anti-normal to orbital plane" mode*
- #define **NAVMODE\_HOLDALT** 7  
*"Hold altitude" mode*
- #define **MANCTRL\_ATTMODE** 0  
*current attitude mode*
- #define **MANCTRL\_REVMODE** 1  
*reverse of current attitude mode*
- #define **MANCTRL\_ROTMODE** 2  
*rotational attitude modes only*
- #define **MANCTRL\_LINMODE** 3

*linear attitude modes only*

- #define **MANCTRL\_ANYMODE** 4

*rotational and linear modes*

- #define **MANCTRL\_KEYBOARD** 0

*keyboard input*

- #define **MANCTRL\_JOYSTICK** 1

*joystick input*

- #define **MANCTRL\_ANYDEVICE** 2

*input from any device*

- #define **COCKPIT\_GENERIC** 1

- #define **COCKPIT\_PANELS** 2

- #define **COCKPIT\_VIRTUAL** 3

- #define **CAM\_COCKPIT** 0

- #define **CAM\_TARGETRELATIVE** 1

- #define **CAM\_ABSDIRECTION** 2

- #define **CAM\_GLOBALFRAME** 3

- #define **CAM\_TARGETTOOBJECT** 4

- #define **CAM\_TARGETFROMOBJECT** 5

- #define **CAM\_GROUNDOBSERVER** 6

- #define **PROP\_ORBITAL** 0x0F

- #define **PROP\_ORBITAL\_ELEMENTS** 0x00

- #define **PROP\_ORBITAL\_FIXEDSTATE** 0x01

- #define **PROP\_ORBITAL\_FIXEDSURF** 0x02

- #define **PROP\_SORBITAL** 0xF0

- #define **PROP\_SORBITAL\_ELEMENTS** (0x0 << 4)

- #define **PROP\_SORBITAL\_FIXEDSTATE** (0x1 << 4)

- #define **PROP\_SORBITAL\_FIXEDSURF** (0x2 << 4)

- #define **PROP\_SORBITAL\_DESTROY** (0x3 << 4)

- #define **USRINPUT\_NEEDANSWER** 1

- #define **RCS\_NONE** 0

*None (RCS off).*

- #define **RCS\_ROT** 1

*Rotational mode.*

- #define **RCS\_LIN** 2

*Linear (translational) mode.*

- #define **HUD\_NONE** 0

*No mode (turn HUD off).*

- #define **HUD\_ORBIT** 1

*Orbit HUD mode.*

- #define **HUD\_SURFACE** 2

*Surface HUD mode.*

- #define **HUD\_DOCKING** 3

*Docking HUD mode.*

- #define **MFD\_REFRESHBUTTONS** -1

*Refresh MFD buttons.*

- #define **MFD\_NONE** 0

*No mode (turn MFD off).*

- #define **MFD\_ORBIT** 1

*Orbit MFD mode.*

- #define **MFD\_SURFACE** 2

*Surface MFD mode.*

- #define **MFD\_MAP** 3

*Map MFD mode.*

- #define **MFD\_HSI** 4

*HSI (horizontal situation indicator) MFD mode.*

- #define **MFD\_LANDING** 5

*VTOL support MFD mode.*

- #define **MFD.Docking** 6

*Docking support MFD mode.*

- #define **MFD\_OPLANEALIGN** 7

*Orbital plane alignment MFD mode.*

- #define **MFD\_OSYNC** 8

*Orbit synchronisation MFD mode.*

- #define **MFD\_TRANSFER** 9

*Transfer orbit MFD mode.*

- #define **MFD\_COMMS** 10

*Communications MFD mode.*

- #define **MFD\_USERTYPE** 64

*User-defined MFD mode.*

- #define **BUILTIN\_MFD\_MODES** 10

*Number of built-in MFD modes.*

- #define **MAXMFD** 10

*Max. number of MFD displays per panel.*

- #define **MFD\_LEFT** 0  
*Left default MFD display.*
- #define **MFD\_RIGHT** 1  
*Right default MFD display.*
- #define **MFD\_USER1** 2  
*User-defined MFD display 1.*
- #define **MFD\_USER2** 3  
*User-defined MFD display 2.*
- #define **MFD\_USER3** 4  
*User-defined MFD display 3.*
- #define **MFD\_USER4** 5  
*User-defined MFD display 4.*
- #define **MFD\_USER5** 6  
*User-defined MFD display 5.*
- #define **MFD\_USER6** 7  
*User-defined MFD display 6.*
- #define **MFD\_USER7** 8  
*User-defined MFD display 7.*
- #define **MFD\_USER8** 9  
*User-defined MFD display 8.*
- #define **PANEL\_LEFT** 0  
*left neighbour*
- #define **PANEL\_RIGHT** 1  
*right neighbour*
- #define **PANEL\_UP** 2  
*above neighbour*
- #define **PANEL\_DOWN** 3  
*below neighbour*
- #define **PANEL\_REDRAW\_NEVER** 0x0000  
*Don't generate redraw events.*
- #define **PANEL\_REDRAW\_ALWAYS** 0x0001  
*Generate event at each frame.*
- #define **PANEL\_REDRAW\_MOUSE** 0x0002  
*Generate event on mouse event.*

- #define **PANEL\_REDRAW\_INIT** 0x0003  
*Initialisation event.*
- #define **PANEL\_REDRAW\_USER** 0x0004  
*User-generated event.*
- #define **PANEL\_REDRAW\_GDI** 0x1000  
*Allow GDI access during redraw events.*
- #define **PANEL\_REDRAW\_SKETCHPAD** 0x2000  
*Allow Sketchpad access during redraw events.*
- #define **PANEL\_MOUSE\_IGNORE** 0x00  
*Don't generate mouse events.*
- #define **PANEL\_MOUSE\_LBUTTONDOWN** 0x01  
*Left button down event.*
- #define **PANEL\_MOUSE\_RBUTTONDOWN** 0x02  
*Right button down event.*
- #define **PANEL\_MOUSE\_LBUTTONUP** 0x04  
*Left button release event.*
- #define **PANEL\_MOUSE\_RBUTTONUP** 0x08  
*Right button release event.*
- #define **PANEL\_MOUSE\_LBAPPED** 0x10  
*Left button down (continuous).*
- #define **PANEL\_MOUSE\_RBAPPED** 0x20  
*Right button down (continuous).*
- #define **PANEL\_MOUSE\_DOWN** 0x03  
*Composite down event.*
- #define **PANEL\_MOUSE\_UP** 0x0C  
*Composite release event.*
- #define **PANEL\_MOUSE\_PRESSED** 0x30  
*Composite down (continuous).*
- #define **PANEL\_MOUSE\_ONREPLAY** 0x40  
*Create mouse events during replay.*
- #define **PANEL\_MAP\_NONE** 0x00  
*area texture is undefined (i.e. should be completely redrawn)*
- #define **PANEL\_MAP\_BACKGROUND** 0x01

- #define **PANEL\_MAP\_CURRENT** 0x02  
*area texture contains a copy of the panel background*
- #define **PANEL\_MAP\_BGONREQUEST** 0x03  
*area texture is undefined, but panel background can be requested*
- #define **PANEL\_ATTACH\_BOTTOM** 0x0001
- #define **PANEL\_ATTACH\_TOP** 0x0002
- #define **PANEL\_ATTACH\_LEFT** 0x0004
- #define **PANEL\_ATTACH\_RIGHT** 0x0008
- #define **PANEL\_MOVEOUT\_BOTTOM** 0x0010
- #define **PANEL\_MOVEOUT\_TOP** 0x0020
- #define **PANEL\_MOVEOUT\_LEFT** 0x0040
- #define **PANEL\_MOVEOUT\_RIGHT** 0x0080
- #define **SURF\_NO\_CK** 0xFFFFFFFF
- #define **SURF\_PREDEF\_CK** 0xFFFFFFFFE
- #define **SURF\_NO\_ROTATION** ((DWORD)-1)
- #define **SURF\_HMIRROR** ((DWORD)-2)
- #define **SURF\_VMIRROR** ((DWORD)-3)
- #define **SURF\_ROTATE\_90** ((DWORD)-4)
- #define **SURF\_ROTATE\_180** ((DWORD)-5)
- #define **SURF\_ROTATE\_270** ((DWORD)-6)
- #define **DLG\_ALLOWMULTI** 0x1
- #define **DLG\_CAPTIONCLOSE** 0x2
- #define **DLG\_CAPTIONHELP** 0x4
- #define **DLG\_CB\_TWOSTATE** 0x1
- #define **OAPI\_MSG\_MFD\_OPENED** 1
- #define **OAPI\_MSG\_MFD\_CLOSED** 2
- #define **OAPI\_MSG\_MFD\_UPDATE** 3
- #define **OAPI\_MSG\_MFD\_OPENEDEX** 4
- #define **VMSG\_LUAINTERPRETER** 0x0001  
*initialise Lua interpreter*
- #define **VMSG\_LUAINSTANCE** 0x0002  
*create Lua vessel instance*
- #define **VMSG\_USER** 0x1000  
*base index for user-defined messages*
- #define **MESHVIS\_NEVER** 0x00  
*Mesh is never visible.*
- #define **MESHVIS\_EXTERNAL** 0x01  
*Mesh is visible in external views.*
- #define **MESHVIS\_COCKPIT** 0x02  
*Mesh is visible in internal (cockpit) views.*

- #define **MESHVIS\_ALWAYS** (MESHVIS\_EXTERNAL|MESHVIS\_COCKPIT)  
*Mesh is always visible.*
- #define **MESHVIS\_VC** 0x04  
*Mesh is only visible in virtual cockpit internal views.*
- #define **MESHVIS\_EXTPASS** 0x10  
*Visibility modifier: render mesh during external pass, even for internal views.*
- #define **MESHPROPERTY\_MODULATEMATALPHA** 1
- #define **TRANSMITTER\_NONE** 0
- #define **TRANSMITTER\_VOR** 1
- #define **TRANSMITTER\_VTOL** 2
- #define **TRANSMITTER\_ILS** 3
- #define **TRANSMITTER\_IDS** 4
- #define **TRANSMITTER\_XPDR** 5
- #define **OBJPRM\_PLANET\_SURFACEMAXLEVEL** 0x0001  
*Max. resolution level for planet surface rendering. (Parameter type: DWORD).*
- #define **OBJPRM\_PLANET\_SURFACERIPPLE** 0x0002  
*Flag for ripple effect on reflective surfaces (Parameter type: bool).*
- #define **OBJPRM\_PLANET\_HAZEEXTENT** 0x0003  
*Bleed-in factor of atmospheric haze into planet disc. (Parameter type: double; range: 0-0.9).*
- #define **OBJPRM\_PLANET\_HAZEDENSITY** 0x0004  
*Density at which the horizon haze is rendered (basic density is calculated from atmospheric density) Default: 1.0. (Parameter type: double).*
- #define **OBJPRM\_PLANET\_HAZESHIFT** 0x0005
- #define **OBJPRM\_PLANET\_HAZECOLOUR** 0x0006
- #define **OBJPRM\_PLANET\_FOGPARAM** 0x0007
- #define **OBJPRM\_PLANET\_SHADOWCOLOUR** 0x0008
- #define **OBJPRM\_PLANET\_HASCLOUDS** 0x0009
- #define **OBJPRM\_PLANET\_CLOUDALT** 0x000A
- #define **OBJPRM\_PLANET\_CLOUDROTATION** 0x000B
- #define **OBJPRM\_PLANET\_CLOUDSHADOWCOL** 0x000C
- #define **OBJPRM\_PLANET\_CLOUDMICROTEX** 0x000D
- #define **OBJPRM\_PLANET\_CLOUDMICROALTMIN** 0x000E
- #define **OBJPRM\_PLANET\_CLOUDMICROALTMAX** 0x000F
- #define **OBJPRM\_PLANET\_HASRINGS** 0x0010
- #define **OBJPRM\_PLANET\_RINGMINRAD** 0x0011
- #define **OBJPRM\_PLANET\_RINGMAXRAD** 0x0012
- #define **OBJPRM\_PLANET\_ATTENUATIONALT** 0x0013  
*Altitude [m] up to which an atmosphere attenuates light cast from the sun on a spacecraft. (Parameter type: double).*
- #define **OAPI\_KEY\_ESCAPE** 0x01  
*Escape key.*

- #define **OAPI\_KEY\_1** 0x02  
*'1' key on main keyboard*
- #define **OAPI\_KEY\_2** 0x03  
*'2' key on main keyboard*
- #define **OAPI\_KEY\_3** 0x04  
*'3' key on main keyboard*
- #define **OAPI\_KEY\_4** 0x05  
*'4' key on main keyboard*
- #define **OAPI\_KEY\_5** 0x06  
*'5' key on main keyboard*
- #define **OAPI\_KEY\_6** 0x07  
*'6' key on main keyboard*
- #define **OAPI\_KEY\_7** 0x08  
*'7' key on main keyboard*
- #define **OAPI\_KEY\_8** 0x09  
*'8' key on main keyboard*
- #define **OAPI\_KEY\_9** 0x0A  
*'9' key on main keyboard*
- #define **OAPI\_KEY\_0** 0x0B  
*'0' key on main keyboard*
- #define **OAPI\_KEY\_MINUS** 0x0C  
*'-' key on main keyboard*
- #define **OAPI\_KEY\_EQUALS** 0x0D  
*'=' key on main keyboard*
- #define **OAPI\_KEY\_BACK** 0x0E  
*backspace key*
- #define **OAPI\_KEY\_TAB** 0x0F  
*tab key*
- #define **OAPI\_KEY\_Q** 0x10  
*'Q' key*
- #define **OAPI\_KEY\_W** 0x11  
*'W' key*
- #define **OAPI\_KEY\_E** 0x12  
*'E' key*

- #define **OAPI\_KEY\_R** 0x13  
*'R' key*
- #define **OAPI\_KEY\_T** 0x14  
*'T' key*
- #define **OAPI\_KEY\_Y** 0x15  
*'Y' key*
- #define **OAPI\_KEY\_U** 0x16  
*'U' key*
- #define **OAPI\_KEY\_I** 0x17  
*'I' key*
- #define **OAPI\_KEY\_O** 0x18  
*'O' key*
- #define **OAPI\_KEY\_P** 0x19  
*'P' key*
- #define **OAPI\_KEY\_LBRACKET** 0x1A  
*'[' (left bracket) key*
- #define **OAPI\_KEY\_RBRACKET** 0x1B  
*']' (right bracket) key*
- #define **OAPI\_KEY\_RETURN** 0x1C  
*'Enter' key on main keyboard*
- #define **OAPI\_KEY\_LCONTROL** 0x1D  
*Left 'Ctrl' key.*
- #define **OAPI\_KEY\_A** 0x1E  
*'A' key*
- #define **OAPI\_KEY\_S** 0x1F  
*'S' key*
- #define **OAPI\_KEY\_D** 0x20  
*'D' key*
- #define **OAPI\_KEY\_F** 0x21  
*'F' key*
- #define **OAPI\_KEY\_G** 0x22  
*'G' key*
- #define **OAPI\_KEY\_H** 0x23

- #define **OAPI\_KEY\_J** 0x24  
*'J' key*
- #define **OAPI\_KEY\_K** 0x25  
*'K' key*
- #define **OAPI\_KEY\_L** 0x26  
*'L' key*
- #define **OAPI\_KEY\_SEMICOLON** 0x27  
*'; (semicolon) key*
- #define **OAPI\_KEY\_APOSTROPHE** 0x28  
*' (apostrophe) key*
- #define **OAPI\_KEY\_GRAVE** 0x29  
*accent grave*
- #define **OAPI\_KEY\_LSHIFT** 0x2A  
*Left 'Shift' key.*
- #define **OAPI\_KEY\_BACKSLASH** 0x2B  
*'\ (Backslash) key*
- #define **OAPI\_KEY\_Z** 0x2C  
*'Z' key*
- #define **OAPI\_KEY\_X** 0x2D  
*'X' key*
- #define **OAPI\_KEY\_C** 0x2E  
*'C' key*
- #define **OAPI\_KEY\_V** 0x2F  
*'V' key*
- #define **OAPI\_KEY\_B** 0x30  
*'B' key*
- #define **OAPI\_KEY\_N** 0x31  
*'N' key*
- #define **OAPI\_KEY\_M** 0x32  
*'M' key*
- #define **OAPI\_KEY\_COMMA** 0x33  
*', (comma) key*

- #define **OAPI\_KEY\_PERIOD** 0x34  
*'.' key on main keyboard*
- #define **OAPI\_KEY\_SLASH** 0x35  
*'/' key on main keyboard*
- #define **OAPI\_KEY\_RSHIFT** 0x36  
*Right 'Shift' key.*
- #define **OAPI\_KEY\_MULTIPLY** 0x37  
*\* on numeric keypad*
- #define **OAPI\_KEY\_LALT** 0x38  
*left Alt*
- #define **OAPI\_KEY\_SPACE** 0x39  
*'Space' key*
- #define **OAPI\_KEY\_CAPITAL** 0x3A  
*caps lock key*
- #define **OAPI\_KEY\_F1** 0x3B  
*F1 function key.*
- #define **OAPI\_KEY\_F2** 0x3C  
*F2 function key.*
- #define **OAPI\_KEY\_F3** 0x3D  
*F3 function key.*
- #define **OAPI\_KEY\_F4** 0x3E  
*F4 function key.*
- #define **OAPI\_KEY\_F5** 0x3F  
*F5 function key.*
- #define **OAPI\_KEY\_F6** 0x40  
*F6 function key.*
- #define **OAPI\_KEY\_F7** 0x41  
*F7 function key.*
- #define **OAPI\_KEY\_F8** 0x42  
*F8 function key.*
- #define **OAPI\_KEY\_F9** 0x43  
*F9 function key.*
- #define **OAPI\_KEY\_F10** 0x44  
*F10 function key.*

- #define **OAPI\_KEY\_NUMLOCK** 0x45  
*'Num Lock' key*
- #define **OAPI\_KEY\_SCROLL** 0x46  
*Scroll lock.*
- #define **OAPI\_KEY\_NUMPAD7** 0x47  
*'7' key on numeric keypad*
- #define **OAPI\_KEY\_NUMPAD8** 0x48  
*'8' key on numeric keypad*
- #define **OAPI\_KEY\_NUMPAD9** 0x49  
*'9' key on numeric keypad*
- #define **OAPI\_KEY\_SUBTRACT** 0x4A  
*'-' key on numeric keypad*
- #define **OAPI\_KEY\_NUMPAD4** 0x4B  
*'4' key on numeric keypad*
- #define **OAPI\_KEY\_NUMPAD5** 0x4C  
*'5' key on numeric keypad*
- #define **OAPI\_KEY\_NUMPAD6** 0x4D  
*'6' key on numeric keypad*
- #define **OAPI\_KEY\_ADD** 0x4E  
*'+' key on numeric keypad*
- #define **OAPI\_KEY\_NUMPAD1** 0x4F  
*'1' key on numeric keypad*
- #define **OAPI\_KEY\_NUMPAD2** 0x50  
*'2' key on numeric keypad*
- #define **OAPI\_KEY\_NUMPAD3** 0x51  
*'3' key on numeric keypad*
- #define **OAPI\_KEY\_NUMPAD0** 0x52  
*'0' key on numeric keypad*
- #define **OAPI\_KEY\_DECIMAL** 0x53  
*'.' key on numeric keypad*
- #define **OAPI\_KEY\_OEM\_102** 0x56  
*| < > on UK/German keyboards*
- #define **OAPI\_KEY\_F11** 0x57

*F11 function key.*

- #define **OAPI\_KEY\_F12** 0x58

*F12 function key.*

- #define **OAPI\_KEY\_NUMPADENTER** 0x9C

*Enter on numeric keypad.*

- #define **OAPI\_KEY\_RCONTROL** 0x9D

*right Control key*

- #define **OAPI\_KEY\_DIVIDE** 0xB5

*'/' key on numeric keypad*

- #define **OAPI\_KEY\_RALT** 0xB8

*right Alt*

- #define **OAPI\_KEY\_HOME** 0xC7

*Home on cursor keypad.*

- #define **OAPI\_KEY\_UP** 0xC8

*up-arrow on cursor keypad*

- #define **OAPI\_KEY\_PRIOR** 0xC9

*PgUp on cursor keypad.*

- #define **OAPI\_KEY\_LEFT** 0xCB

*left-arrow on cursor keypad*

- #define **OAPI\_KEY\_RIGHT** 0xCD

*right-arrow on cursor keypad*

- #define **OAPI\_KEY\_END** 0xCF

*End on cursor keypad.*

- #define **OAPI\_KEY\_DOWN** 0xD0

*down-arrow on cursor keypad*

- #define **OAPI\_KEY\_NEXT** 0xD1

*PgDn on cursor keypad.*

- #define **OAPI\_KEY\_INSERT** 0xD2

*Insert on cursor keypad.*

- #define **OAPI\_KEY\_DELETE** 0xD3

*Delete on cursor keypad.*

- #define **KEYDOWN**(buf, key) (buf[key] & 0x80)

- #define **RESETKEY**(buf, key) (buf[key] = 0)

- #define **KEYMOD\_LSHIFT**(buf) (KEYDOWN(buf,OAPI\_KEY\_LSHIFT))

- #define **KEYMOD\_RSHIFT**(buf) (KEYDOWN(buf,OAPI\_KEY\_RSHIFT))
- #define **KEYMOD\_SHIFT**(buf) (KEYMOD\_LSHIFT(buf) || KEYMOD\_RSHIFT(buf))
- #define **KEYMOD\_LCONTROL**(buf) (KEYDOWN(buf,OAPI\_KEY\_LCONTROL))
- #define **KEYMOD\_RCONTROL**(buf) (KEYDOWN(buf,OAPI\_KEY\_RCONTROL))
- #define **KEYMOD\_CONTROL**(buf) (KEYMOD\_LCONTROL(buf) || KEYMOD\_RCONTROL(buf))
- #define **KEYMOD\_LALT**(buf) (KEYDOWN(buf,OAPI\_KEY\_LALT))
- #define **KEYMOD\_RALT**(buf) (KEYDOWN(buf,OAPI\_KEY\_RALT))
- #define **KEYMOD\_ALT**(buf) (KEYMOD\_LALT(buf) || KEYMOD\_RALT(buf))
- #define **OAPI\_LKEY\_CockpitRotateLeft** 0
  - rotate camera left in cockpit view*
- #define **OAPI\_LKEY\_CockpitRotateRight** 1
  - rotate camera right in cockpit view*
- #define **OAPI\_LKEY\_CockpitRotateUp** 2
  - rotate camera up in cockpit view*
- #define **OAPI\_LKEY\_CockpitRotateDown** 3
  - rotate camera down in cockpit view*
- #define **OAPI\_LKEY\_CockpitDontLean** 4
  - return to default cockpit camera position*
- #define **OAPI\_LKEY\_CockpitLeanForward** 5
  - move cockpit camera forward*
- #define **OAPI\_LKEY\_CockpitLeanLeft** 6
  - move cockpit camera left*
- #define **OAPI\_LKEY\_CockpitLeanRight** 7
  - move cockpit camera right*
- #define **OAPI\_LKEY\_CockpitResetCam** 8
  - rotate and shift cockpit camera back to default*
- #define **OAPI\_LKEY\_PanelShiftLeft** 9
  - shift 2D instrument panel left*
- #define **OAPI\_LKEY\_PanelShiftRight** 10
  - shift 2D instrument panel right*
- #define **OAPI\_LKEY\_PanelShiftUp** 11
  - shift 2D instrument panel up*
- #define **OAPI\_LKEY\_PanelShiftDown** 12
  - shift 2D instrument panel down*
- #define **OAPI\_LKEY\_PanelSwitchLeft** 13
  - switch to left neighbour panel*

- #define **OAPI\_LKEY\_PanelSwitchRight** 14  
*switch to right neighbour panel*
- #define **OAPI\_LKEY\_PanelSwitchUp** 15  
*switch to upper neighbour panel*
- #define **OAPI\_LKEY\_PanelSwitchDown** 16  
*switch to lower neighbour panel*
- #define **OAPI\_LKEY\_TrackRotateLeft** 17  
*turn track view camera left*
- #define **OAPI\_LKEY\_TrackRotateRight** 18  
*turn track view camera right*
- #define **OAPI\_LKEY\_TrackRotateUp** 19  
*turn track view camera up*
- #define **OAPI\_LKEY\_TrackRotateDown** 20  
*turn track view camera down*
- #define **OAPI\_LKEY\_TrackAdvance** 21  
*advance track view camera towards target*
- #define **OAPI\_LKEY\_TrackRetreat** 22  
*retreat track view camera from target*
- #define **OAPI\_LKEY\_GroundTiltLeft** 23  
*tilt camera left in ground view*
- #define **OAPI\_LKEY\_GroundTiltRight** 24  
*tilt camera right in ground view*
- #define **OAPI\_LKEY\_GroundTiltUp** 25  
*tilt camera up in ground view*
- #define **OAPI\_LKEY\_GroundTiltDown** 26  
*tilt camera down in ground view*
- #define **OAPI\_LKEY\_IncMainThrust** 27  
*increment thrust of main thrusters*
- #define **OAPI\_LKEY\_DecMainThrust** 28  
*decrement thrust of main thrusters*
- #define **OAPI\_LKEY\_KillMainRetro** 29  
*kill main and retro thrusters*
- #define **OAPI\_LKEY\_FullMainThrust** 30

- `#define OAPI_LKEY_FullRetroThrust 31`  
*temporary full main thrust*
- `#define OAPI_LKEY_IncHoverThrust 32`  
*increment thrust of hover thrusters*
- `#define OAPI_LKEY_DecHoverThrust 33`  
*decrement thrust of hover thrusters*
- `#define OAPI_LKEY_RCSEnable 34`  
*enable/disable RCS (reaction control system)*
- `#define OAPI_LKEY_RCSMode 35`  
*toggle linear/rotational RCS mode*
- `#define OAPI_LKEY_RCSPitchUp 36`  
*rotational RCS: pitch up*
- `#define OAPI_LKEY_RCSPitchDown 37`  
*rotational RCS: pitch down*
- `#define OAPI_LKEY_RCSYawLeft 38`  
*rotational RCS: yaw left*
- `#define OAPI_LKEY_RCSYawRight 39`  
*rotational RCS: yaw right*
- `#define OAPI_LKEY_RCSBankLeft 40`  
*rotational RCS: bank left*
- `#define OAPI_LKEY_RCSBankRight 41`  
*rotational RCS: bank right*
- `#define OAPI_LKEY_RCSUp 42`  
*linear RCS: accelerate up (+y)*
- `#define OAPI_LKEY_RCSDown 43`  
*linear RCS: accelerate down (-y)*
- `#define OAPI_LKEY_RCSLeft 44`  
*linear RCS: accelerate left (-x)*
- `#define OAPI_LKEY_RCSRRight 45`  
*linear RCS: accelerate right (+x)*
- `#define OAPI_LKEY_RCSForward 46`  
*linear RCS: accelerate forward (+z)*

- #define **OAPI\_LKEY\_RCSBack** 47  
*linear RCS: accelerate backward (-z)*
- #define **OAPI\_LKEY\_LPRCSPitchUp** 48  
*rotational RCS: pitch up 10%*
- #define **OAPI\_LKEY\_LPRCSPitchDown** 49  
*rotational RCS: pitch down 10%*
- #define **OAPI\_LKEY\_LPRCSYawLeft** 50  
*rotational RCS: yaw left 10%*
- #define **OAPI\_LKEY\_LPRCSYawRight** 51  
*rotational RCS: yaw right 10%*
- #define **OAPI\_LKEY\_LPRCSBankLeft** 52  
*rotational RCS: bank left 10%*
- #define **OAPI\_LKEY\_LPRCSBankRight** 53  
*rotational RCS: bank right 10%*
- #define **OAPI\_LKEY\_LPRCSUp** 54  
*linear RCS: accelerate up 10% (+y)*
- #define **OAPI\_LKEY\_LPRCSDown** 55  
*linear RCS: accelerate down 10% (-y)*
- #define **OAPI\_LKEY\_LPRCSLeft** 56  
*linear RCS: accelerate left 10% (-x)*
- #define **OAPI\_LKEY\_LPRCSRRight** 57  
*linear RCS: accelerate right 10% (+x)*
- #define **OAPI\_LKEY\_LPRCSForward** 58  
*linear RCS: accelerate forward 10% (+z)*
- #define **OAPI\_LKEY\_LPRCSBack** 59  
*linear RCS: accelerate backward 10% (-z)*
- #define **OAPI\_LKEY\_NMHoldAltitude** 60  
*toggle navmode: hold altitude*
- #define **OAPI\_LKEY\_NMHLevel** 61  
*toggle navmode: level with horizon*
- #define **OAPI\_LKEY\_NMPrograde** 62  
*toggle navmode: prograde*
- #define **OAPI\_LKEY\_NMRetrograde** 63  
*toggle navmode: retrograde*

- #define **OAPI\_LKEY\_NMNormal** 64  
*toggle navmode: normal to orbital plane*
- #define **OAPI\_LKEY\_NMAntinormal** 65  
*toggle navmode: antinormal to orbital plane*
- #define **OAPI\_LKEY\_NMKillrot** 66  
*toggle navmode: kill rotation*
- #define **OAPI\_LKEY\_Undock** 67  
*undock from docked vessel*
- #define **OAPI\_LKEY\_IncElevatorTrim** 68  
*increment elevator trim setting*
- #define **OAPI\_LKEY\_DecElevatorTrim** 69  
*decrement elevator trim setting*
- #define **OAPI\_LKEY\_WheelbrakeLeft** 70  
*apply wheelbrake left*
- #define **OAPI\_LKEY\_WheelbrakeRight** 71  
*apply wheelbrake right*
- #define **OAPI\_LKEY\_HUD** 72  
*toggle HUD on/off*
- #define **OAPI\_LKEY\_HUDMode** 73  
*switch through HUD modes*
- #define **OAPI\_LKEY\_HUDReference** 74  
*query reference object for HUD display*
- #define **OAPI\_LKEY\_HUDTarget** 75  
*query target object for HUD display*
- #define **OAPI\_LKEY\_HUDColour** 76  
*switch through HUD colours*
- #define **OAPI\_LKEY\_IncSimSpeed** 77  
*increase simulation speed x10*
- #define **OAPI\_LKEY\_DecSimSpeed** 78  
*decrease simulation speed x0.1*
- #define **OAPI\_LKEY\_IncFOV** 79  
*increment field of view*
- #define **OAPI\_LKEY\_DecFOV** 80

*decrement field of view*

- #define **OAPI\_LKEY\_StepIncFOV** 81  
*increment field of view by 10 deg*
- #define **OAPI\_LKEY\_StepDecFOV** 82  
*decrement field of view by 10 deg*
- #define **OAPI\_LKEY\_MainMenu** 83  
*open main menu*
- #define **OAPI\_LKEY\_DlgHelp** 84  
*open help dialog*
- #define **OAPI\_LKEY\_DlgCamera** 85  
*open camera dialog*
- #define **OAPI\_LKEY\_DlgSimspeed** 86  
*open simulation speed dialog*
- #define **OAPI\_LKEY\_DlgCustomCmd** 87  
*open custom command dialog*
- #define **OAPI\_LKEY\_DlgVisHelper** 88  
*open visual helper dialog*
- #define **OAPI\_LKEY\_DlgRecorder** 89  
*open flight recorder dialog*
- #define **OAPI\_LKEY\_DlgInfo** 90  
*open object info dialog*
- #define **OAPI\_LKEY\_DlgMap** 91  
*open map dialog*
- #define **OAPI\_LKEY\_DlgNavaid** 92  
*open nav transmitter list*
- #define **OAPI\_LKEY\_ToggleInfo** 93  
*OBSOLETE.*
- #define **OAPI\_LKEY\_ToggleFPS** 94  
*OBSOLETE.*
- #define **OAPI\_LKEY\_ToggleCamInternal** 95  
*switch between cockpit and external camera*
- #define **OAPI\_LKEY\_ToggleTrackMode** 96  
*switch between track camera modes*

- #define **OAPI\_LKEY\_TogglePanelMode** 97  
*switch between cockpit modes*
- #define **OAPI\_LKEY\_TogglePlanetarium** 98  
*toggle celestial marker display on/off*
- #define **OAPI\_LKEY\_ToggleRecPlay** 99  
*toggle flight recorder/playback on/off*
- #define **OAPI\_LKEY\_Pause** 100  
*toggle simulation pause on/off*
- #define **OAPI\_LKEY\_Quicksave** 101  
*quick-save current simulation state*
- #define **OAPI\_LKEY\_Quit** 102  
*quit simulation session*
- #define **OAPI\_LKEY\_DlgSelectVessel** 103  
*open vessel selection dialog*
- #define **OAPI\_LKEY\_SelectPrevVessel** 104  
*switch focus to previous vessel*
- #define **LKEY\_COUNT** 105  
*number of logical key definitions*

## Typedefs

- typedef void \* **OBJHANDLE**  
*Handle for objects (vessels, stations, planets).*
- typedef void \* **VISHANDLE**  
*Handle for visuals.*
- typedef void \* **MESHHANDLE**  
*Handle for meshes.*
- typedef int \* **DEVMESHHANDLE**  
*Handle for graphics-client-specific meshes.*
- typedef void \* **SURFHANDLE**  
*Handle for bitmap surfaces and textures (panels and panel items).*
- typedef void \* **PANELHANDLE**  
*Handle for 2D instrument panels.*
- typedef void \* **FILEHANDLE**

*Handle for file streams.*

- `typedef void * INTERPRETERHANDLE`

*Handle for script interpreters.*

- `typedef void * THRUSTER_HANDLE`

*Handle for thrusters.*

- `typedef void * THGROUP_HANDLE`

*Handle for logical thruster groups.*

- `typedef void * PROPELLANT_HANDLE`

*Propellant resource handle.*

- `typedef void * PSTREAM_HANDLE`

*Handle for particle streams.*

- `typedef void * DOCKHANDLE`

*Handle for vessel docking ports.*

- `typedef void * ATTACHMENTHANDLE`

*Handle for vessel passive attachment points.*

- `typedef void * AIRFOILHANDLE`

*Handle for vessel airfoils.*

- `typedef void * CTRLSURFHANDLE`

*Handle for vessel aerodynamic control surfaces.*

- `typedef void * NAVHANDLE`

*Handle for a navigation radio transmitter (VOR, ILS, IDS, XPDR).*

- `typedef void * ANIMATIONCOMPONENT_HANDLE`

*Handle for animation components.*

- `typedef void * LAUNCHPADITEM_HANDLE`

*Handle for custom items added to Launchpad "Extra" list.*

- `typedef void * NOTEHANDLE`

*Handle for onscreen annotation objects.*

- `typedef bool(* Listentry_clbk )(char *name, DWORD idx, DWORD flag, void *usrdata)`

*Callback function for list entry selections.*

- `typedef double(* LiftCoeffFunc )(double aoa)`

- `typedef void(* AirfoilCoeffFunc )(double aoa, double M, double Re, double *cl, double *cm, double *cd)`

- `typedef void(* AirfoilCoeffFuncEx )(VESSEL *v, double aoa, double M, double Re, void *context, double *cl, double *cm, double *cd)`

- `typedef int(* KeyFunc )(const char *keybuf)`

- **typedef void(\* LoadMeshClbkFunc )(MESHHANDLE hMesh, bool firstload)**  
*Callback function used by `oapiLoadMeshGlobal(const char*, LoadMeshClbkFunc)`.*
- **typedef void(\* CustomFunc )(void \*context)**

## Enumerations

- **enum FileAccessMode { FILE\_IN, FILE\_OUT, FILE\_APP }**
- **enum PathRoot {**  
**ROOT, CONFIG, SCENARIOS, TEXTURES,**  
**TEXTURES2, MESHES, MODULES }**
- **enum REFFRAME { FRAME\_GLOBAL, FRAME\_LOCAL, FRAME\_REFLOCAL, FRAME\_HORIZON }**  
*Identifiers for frames of reference.*
- **enum ENGINETYPE { ENGINE\_MAIN, ENGINE\_RETRO, ENGINE\_HOVER, ENGINE\_ATTITUDE }**  
*Thruster group identifiers (obsolete).*
- **enum EXHAUSTTYPE { EXHAUST\_MAIN, EXHAUST\_RETRO, EXHAUST\_HOVER, EXHAUST\_CUSTOM }**
- **enum THGROUP\_TYPE {**  
**THGROUP\_MAIN, THGROUP\_RETRO, THGROUP\_HOVER, THGROUP\_ATT\_PITCHUP,**  
**THGROUP\_ATT\_PITCHDOWN, THGROUP\_ATT\_YAWLEFT, THGROUP\_ATT\_YAWRIGHT,**  
**THGROUP\_ATT\_BANKLEFT,**  
**THGROUP\_ATT\_BANKRIGHT, THGROUP\_ATT\_RIGHT, THGROUP\_ATT\_LEFT,**  
**THGROUP\_ATT\_UP,**  
**THGROUP\_ATT\_DOWN, THGROUP\_ATT\_FORWARD, THGROUP\_ATT\_BACK,**  
**THGROUP\_USER = 0x40 }**  
*Thruster group types.*
- **enum ATTITUDEMODE { ATTMODE\_DISABLED, ATTMODE\_ROT, ATTMODE\_LIN }**
- **enum AIRFOIL\_ORIENTATION { LIFT\_VERTICAL, LIFT\_HORIZONTAL }**  
*Lift vector orientation for airfoils.*
- **enum AIRCTRL\_TYPE {**  
**AIRCTRL\_ELEVATOR, AIRCTRL\_RUDDER, AIRCTRL\_AILERON, AIRCTRL\_FLAP,**  
**AIRCTRL\_ELEVATORTRIM, AIRCTRL\_RUDDERTRIM }**  
*Control surfaces provide attitude and drag control during atmospheric flight.*
- **enum FontStyle { FONT\_NORMAL = 0, FONT\_BOLD = 1, FONT\_ITALIC = 2, FONT\_UNDERLINE = 4 }**

## Functions

- **double normangle (double angle)**  
*Returns the input argument normalised to range -pi ... pi.*

- double [posangle](#) (double angle)  
*Returns the input argument normalised to range 0 ... 2 pi.*
- OAPIFUNC int [oapiGetOrbiterVersion](#) ()  
*Returns the version number of the Orbiter core system.*
- int [oapiGetModuleVersion](#) ()  
*Returns the API version number against which the module was linked.*
- OAPIFUNC HINSTANCE [oapiGetOrbiterInstance](#) ()  
*Returns the instance handle for the running Orbiter application.*
- OAPIFUNC const char \* [oapiGetCmdLine](#) ()  
*Returns a pointer to the command line with which Orbiter was invoked.*
- OAPIFUNC void [oapiGetViewportSize](#) (DWORD \*w, DWORD \*h, DWORD \*bpp=0)  
*Returns the dimensions of the render viewport.*
- OAPIFUNC double [oapiGetPanelScale](#) ()  
*Returns the scaling factor for 2-D instrument panels.*
- OAPIFUNC void [oapiRegisterModule](#) (oapi::Module \*module)  
*Register a module interface class instance.*
- OAPIFUNC char \* [oapiDebugString](#) ()  
*Returns a pointer to a string which will be displayed in the lower left corner of the viewport.*
- OAPIFUNC OBJHANDLE [oapiGetObjectByName](#) (char \*name)  
*Returns a handle for a named simulation object.*
- OAPIFUNC OBJHANDLE [oapiGetObjectByIndex](#) (int index)  
*Returns a handle for an indexed simulation object.*
- OAPIFUNC DWORD [oapiGetObjectCount](#) ()  
*Returns the number of objects currently present in the simulation.*
- OAPIFUNC int [oapiGetObjectType](#) (OBJHANDLE hObj)  
*Returns the type of an object identified by its handle.*
- OAPIFUNC const void \* [oapiGetObjectParam](#) (OBJHANDLE hObj, DWORD paramtype)  
*Returns an object-specific configuration parameter.*
- OAPIFUNC OBJHANDLE [oapiGetVesselByName](#) (char \*name)  
*Returns the handle of a vessel identified by its name.*
- OAPIFUNC OBJHANDLE [oapiGetVesselByIndex](#) (int index)  
*Returns the handle of a vessel identified by its reference index.*
- OAPIFUNC DWORD [oapiGetVesselCount](#) ()  
*Returns the number of vessels currently present in the simulation.*

- OAPIFUNC bool [oapiIsVessel](#) (**OBJHANDLE** hVessel)  
*Checks if the specified handle is a valid vessel handle.*
- OAPIFUNC **OBJHANDLE** [oapiGetGbodyByName](#) (char \*name)  
*Returns the handle of a celestial body (sun, planet or moon) identified by its name.*
- OAPIFUNC **OBJHANDLE** [oapiGetGbodyByIndex](#) (int index)  
*Returns the handle of a celestial body (sun, planet or moon) indentified by its list index.*
- OAPIFUNC **DWORD** [oapiGetGbodyCount](#) ()  
*Returns the number of celestial bodies (sun, planets and moons) currently present in the simulation.*
- OAPIFUNC **OBJHANDLE** [oapiGetBaseByName](#) (**OBJHANDLE** hPlanet, char \*name)  
*Returns the handle of a surface base on a given planet or moon.*
- OAPIFUNC **OBJHANDLE** [oapiGetBaseByIndex](#) (**OBJHANDLE** hPlanet, int index)  
*Returns the handle of a surface base on a planet or moon given by its list index.*
- OAPIFUNC **DWORD** [oapiGetBaseCount](#) (**OBJHANDLE** hPlanet)  
*Returns the number of surface bases defined for a given planet.*
- OAPIFUNC void [oapiGetObjectName](#) (**OBJHANDLE** hObj, char \*name, int n)  
*Returns the name of an object.*
- OAPIFUNC **OBJHANDLE** [oapiGetFocusObject](#) ()  
*Returns the handle for the current focus object.*
- OAPIFUNC **OBJHANDLE** [oapiSetFocusObject](#) (**OBJHANDLE** hVessel)  
*Switches the input focus to a different vessel object.*
- OAPIFUNC **VESSEL** \* [oapiGetVesselInterface](#) (**OBJHANDLE** hVessel)  
*Returns a **VESSEL** class instance for a vessel.*
- OAPIFUNC **VESSEL** \* [oapiGetFocusInterface](#) ()  
*Returns the **VESSEL** class instance for the current focus object.*
- OAPIFUNC **CELBODY** \* [oapiGetCelbodyInterface](#) (**OBJHANDLE** hBody)  
*Returns a **CELBODY** interface instance for a celestial body, if available.*
- OAPIFUNC **OBJHANDLE** [oapiCreateVessel](#) (const char \*name, const char \*classname, const **VESSELSTATUS** &status)  
*Creates a new vessel.*
- OAPIFUNC **OBJHANDLE** [oapiCreateVesselEx](#) (const char \*name, const char \*classname, const void \*status)  
*Creates a new vessel via a VESSELSTATUSx (x >= 2) interface.*
- OAPIFUNC bool [oapiDeleteVessel](#) (**OBJHANDLE** hVessel, **OBJHANDLE** hAlternativeCameraTarget=0)  
*Deletes a vessel.*

*Deletes an existing vessel.*

- OAPIFUNC void `oapiGetBarycentre (OBJHANDLE hObj, VECTOR3 *bary)`  
*Returns the global position of the barycentre of a complete planetary system or a single planet-moons system.*
- OAPIFUNC double `oapiGetSize (OBJHANDLE hObj)`  
*Returns the size (mean radius) of an object.*
- OAPIFUNC double `oapiGetMass (OBJHANDLE hObj)`  
*Returns the mass of an object. For vessels, this is the total mass, including current fuel mass.*
- OAPIFUNC void `oapiGetGlobalPos (OBJHANDLE hObj, VECTOR3 *pos)`  
*Returns the position of an object in the global reference frame.*
- OAPIFUNC void `oapiGetGlobalVel (OBJHANDLE hObj, VECTOR3 *vel)`  
*Returns the velocity of an object in the global reference frame.*
- OAPIFUNC void `oapiGetRelativePos (OBJHANDLE hObj, OBJHANDLE hRef, VECTOR3 *pos)`  
*Returns the distance vector from hRef to hObj in the ecliptic reference frame.*
- OAPIFUNC void `oapiGetRelativeVel (OBJHANDLE hObj, OBJHANDLE hRef, VECTOR3 *vel)`  
*Returns the velocity difference vector of hObj relative to hRef in the ecliptic reference frame.*
- OAPIFUNC double `oapiGetEmptyMass (OBJHANDLE hVessel)`  
*Returns empty mass of a vessel, excluding fuel.*
- OAPIFUNC void `oapiSetEmptyMass (OBJHANDLE hVessel, double mass)`  
*Set the empty mass of a vessel (excluding fuel).*
- OAPIFUNC double `oapiGetFuelMass (OBJHANDLE hVessel)`  
*Returns current fuel mass of the first propellant resource of a vessel.*
- OAPIFUNC double `oapiGetMaxFuelMass (OBJHANDLE hVessel)`  
*Returns maximum fuel capacity of the first propellant resource of a vessel.*
- OAPIFUNC PROPELLANT\_HANDLE `oapiGetPropellantHandle (OBJHANDLE hVessel, DWORD idx)`  
*Returns an identifier of a vessel's propellant resource.*
- OAPIFUNC double `oapiGetPropellantMass (PROPELLANT_HANDLE ph)`  
*Returns the current fuel mass [kg] of a propellant resource.*
- OAPIFUNC double `oapiGetPropellantMaxMass (PROPELLANT_HANDLE ph)`  
*Returns the maximum capacity [kg] of a propellant resource.*
- OAPIFUNC DOCKHANDLE `oapiGetDockHandle (OBJHANDLE hVessel, UINT n)`  
*Returns a handle to a vessel docking port.*

- OAPIFUNC **OBJHANDLE oapiGetDockStatus (DOCKHANDLE dock)**  
*Returns the handle of a vessel docked at a port.*
- OAPIFUNC void **oapiGetFocusGlobalPos (VECTOR3 \*pos)**  
*Returns the position of the current focus object in the global reference frame.*
- OAPIFUNC void **oapiGetFocusGlobalVel (VECTOR3 \*vel)**  
*Returns the velocity of the current focus object in the global reference frame.*
- OAPIFUNC void **oapiGetFocusRelativePos (OBJHANDLE hRef, VECTOR3 \*pos)**  
*Returns the distance vector from hRef to the current focus object.*
- OAPIFUNC void **oapiGetFocusRelativeVel (OBJHANDLE hRef, VECTOR3 \*vel)**  
*Returns the velocity difference vector of the current focus object relative to hRef.*
- OAPIFUNC BOOL **oapiGetAltitude (OBJHANDLE hVessel, double \*alt)**  
*Returns the altitude of a vessel over a planetary surface.*
- OAPIFUNC BOOL **oapiGetPitch (OBJHANDLE hVessel, double \*pitch)**  
*Returns a vessel's pitch angle w.r.t. the local horizon.*
- OAPIFUNC BOOL **oapiGetBank (OBJHANDLE hVessel, double \*bank)**  
*Returns a vessel's bank angle w.r.t. the local horizon.*
- OAPIFUNC BOOL **oapiGetHeading (OBJHANDLE hVessel, double \*heading)**  
*Returns a vessel's heading (against geometric north) calculated for the local horizon plane.*
- OAPIFUNC BOOL **oapiGetFocusAltitude (double \*alt)**  
*Returns the altitude of the current focus vessel over a planetary surface.*
- OAPIFUNC BOOL **oapiGetFocusPitch (double \*pitch)**  
*Returns the pitch angle of the current focus vessel w.r.t. the local horizon.*
- OAPIFUNC BOOL **oapiGetFocusBank (double \*bank)**  
*Returns the bank angle of the current focus vessel w.r.t. the local horizon.*
- OAPIFUNC BOOL **oapiGetFocusHeading (double \*heading)**  
*Returns the heading (against geometric north) of the current focus vessel calculated for the local horizon plane.*
- OAPIFUNC BOOL **oapiGetGroundspeed (OBJHANDLE hVessel, double \*groundspeed)**  
*Returns a vessel's ground speed w.r.t. the closest planet or moon.*
- OAPIFUNC bool **oapiGetGroundspeedVector (OBJHANDLE hVessel, REFFRAME frame, VECTOR3 \*vel)**  
*Returns a vessel's groundspeed vector w.r.t. the closest planet or moon in the requested frame of reference.*
- OAPIFUNC BOOL **oapiGetAirspeed (OBJHANDLE hVessel, double \*airspeed)**  
*Returns a vessel's true airspeed w.r.t. the closest planet or moon.*

- OAPIFUNC bool `oapiGetAirspeedVector` (`OBJHANDLE hVessel`, `REFFRAME frame`, `VECTOR3 *v`)

*Returns a vessel's true airspeed vector w.r.t. the closest planet or moon in the requested frame of reference.*
- OAPIFUNC BOOL `oapiGetEquPos` (`OBJHANDLE hVessel`, `double *longitude`, `double *latitude`, `double *radius`)

*Returns a vessel's spherical equatorial coordinates (longitude, latitude and radius) with respect to the closest planet or moon.*
- OAPIFUNC BOOL `oapiGetFocusEquPos` (`double *longitude`, `double *latitude`, `double *radius`)

*Returns the current focus vessel's spherical equatorial coordinates (longitude, latitude and radius) with respect to the closest planet or moon.*
- OAPIFUNC void `oapiGetAtm` (`OBJHANDLE hVessel`, `ATMPARAM *prm`, `OBJHANDLE *hAtmRef=0`)

*Returns the atmospheric parameters at the current vessel position.*
- OAPIFUNC void `oapiGetEngineStatus` (`OBJHANDLE hVessel`, `ENGINESTATUS *es`)

*Retrieve the status of main, retro and hover thrusters for a vessel.*
- OAPIFUNC void `oapiGetFocusEngineStatus` (`ENGINESTATUS *es`)

*Retrieve the engine status for the focus vessel.*
- OAPIFUNC void `oapiSetEngineLevel` (`OBJHANDLE hVessel`, `ENGINETYPE engine`, `double level`)

*Engage the specified engines.*
- OAPIFUNC int `oapiGetAttitudeMode` (`OBJHANDLE hVessel`)

*Returns a vessel's current attitude thruster mode.*
- OAPIFUNC int `oapiToggleAttitudeMode` (`OBJHANDLE hVessel`)

*Flip a vessel's attitude thruster mode between rotational and linear.*
- OAPIFUNC bool `oapiSetAttitudeMode` (`OBJHANDLE hVessel`, `int mode`)

*Set a vessel's attitude thruster mode.*
- OAPIFUNC int `oapiGetFocusAttitudeMode` ()

*Returns the current focus vessel's attitude thruster mode (rotational or linear).*
- OAPIFUNC int `oapiToggleFocusAttitudeMode` ()

*Flip the current focus vessel's attitude thruster mode between rotational and linear.*
- OAPIFUNC bool `oapiSetFocusAttitudeMode` (`int mode`)

*Set the current focus vessel's attitude thruster mode.*
- OAPIFUNC void `oapiGetRotationMatrix` (`OBJHANDLE hObj`, `MATRIX3 *mat`)

*Returns the current rotation matrix of an object.*
- OAPIFUNC void `oapiGlobalToLocal` (`OBJHANDLE hObj`, `const VECTOR3 *glob`, `VECTOR3 *loc`)

*Maps a point from the global frame to a local object frame.*

- OAPIFUNC void [oapiLocalToGlobal](#) (OBJHANDLE hObj, const VECTOR3 \*loc, VECTOR3 \*glob)

*Maps a point from a local object frame to the global frame.*

- OAPIFUNC void [oapiEquToLocal](#) (OBJHANDLE hObj, double lng, double lat, double rad, VECTOR3 \*loc)

*Returns the cartesian position in the local object frame of a point given in equatorial coordinates.*

- OAPIFUNC void [oapiLocalToEqu](#) (OBJHANDLE hObj, const VECTOR3 &loc, double \*lng, double \*lat, double \*rad)

*Returns the equatorial coordinates of a point given in the local frame of an object.*

- OAPIFUNC void [oapiEquToGlobal](#) (OBJHANDLE hObj, double lng, double lat, double rad, VECTOR3 \*glob)

*Returns the global cartesian position of a point given in equatorial coordinates of an object.*

- OAPIFUNC void [oapiGlobalToEqu](#) (OBJHANDLE hObj, const VECTOR3 &glob, double \*lng, double \*lat, double \*rad)

*Returns the equatorial coordinates with respect to an object of a point given in the global reference frame.*

- OAPIFUNC double [oapiOrthodome](#) (double lng1, double lat1, double lng2, double lat2)

*Returns the angular distance of two points on a sphere.*

- OAPIFUNC SURFHANDLE [oapiRegisterExhaustTexture](#) (char \*name)

*Request a custom texture for vessel exhaust rendering.*

- OAPIFUNC SURFHANDLE [oapiRegisterReentryTexture](#) (char \*name)

*Request a custom texture for vessel reentry flame rendering.*

- OAPIFUNC SURFHANDLE [oapiRegisterParticleTexture](#) (char \*name)

- OAPIFUNC void [oapiSetShowGrapplePoints](#) (bool show)

- OAPIFUNC bool [oapiGetShowGrapplePoints](#) ()

- OAPIFUNC double [oapiGetInducedDrag](#) (double cl, double A, double e)

*Aerodynamics helper function.*

- OAPIFUNC double [oapiGetWaveDrag](#) (double M, double M1, double M2, double M3, double cmax)

*Aerodynamics helper function.*

- OAPIFUNC bool [oapiCameraInternal](#) ()

*Returns flag to indicate internal/external camera mode.*

- OAPIFUNC int [oapiCameraMode](#) ()

*Returns the current camera view mode.*

- OAPIFUNC int [oapiCockpitMode](#) ()

*Returns the current cockpit display mode.*

- OAPIFUNC **OBJHANDLE oapiCameraTarget ()**  
*Returns a handle to the current camera target.*
- OAPIFUNC **OBJHANDLE oapiCameraProxyGbody ()**  
*Returns celestial body whose surface is closest to the camera.*
- OAPIFUNC void **oapiCameraGlobalPos (VECTOR3 \*gpos)**  
*Returns current camera position in global coordinates.*
- OAPIFUNC void **oapiCameraGlobalDir (VECTOR3 \*gdir)**  
*Returns current camera direction in global coordinates.*
- OAPIFUNC void **oapiCameraRotationMatrix (MATRIX3 \*rmat)**
- OAPIFUNC double **oapiCameraTargetDist ()**  
*Returns the distance between the camera and its target [m].*
- OAPIFUNC double **oapiCameraAzimuth ()**  
*Returns the current camera azimuth angle with respect to the target.*
- OAPIFUNC double **oapiCameraPolar ()**  
*Returns the current camera polar angle with respect to the target.*
- OAPIFUNC double **oapiCameraAperture ()**  
*Returns the current camera aperture (the field of view) in rad.*
- OAPIFUNC void **oapiCameraSetAperture (double aperture)**  
*Change the camera aperture (field of view).*
- OAPIFUNC void **oapiCameraScaleDist (double dscale)**  
*Moves the camera closer to the target or further away.*
- OAPIFUNC void **oapiCameraRotAzimuth (double dazimuth)**  
*Rotate the camera around the target (azimuth angle).*
- OAPIFUNC void **oapiCameraRotPolar (double dpolar)**  
*Rotate the camera around the target (polar angle).*
- OAPIFUNC void **oapiCameraSetCockpitDir (double polar, double azimuth, bool transition=false)**  
*Set the camera direction in cockpit mode.*
- OAPIFUNC void **oapiCameraAttach (OBJHANDLE hObj, int mode)**  
*Attach the camera to a new target, or switch between internal and external camera mode.*
- OAPIFUNC double **oapiGetPlanetPeriod (OBJHANDLE hPlanet)**  
*Returns the rotation period (the length of a sidereal day) of a planet.*
- OAPIFUNC double **oapiGetPlanetObliquity (OBJHANDLE hPlanet)**  
*Returns the obliquity of the planet's rotation axis (the angle between the rotation axis and the ecliptic zenith).*

- OAPIFUNC double [oapiGetPlanetTheta](#) (**OBJHANDLE** hPlanet)  
*Returns the longitude of the ascending node.*
- OAPIFUNC void [oapiGetPlanetObliquityMatrix](#) (**OBJHANDLE** hPlanet, **MATRIX3** \*mat)  
*Returns a rotation matrix which performs the transformation from the planet's tilted coordinates into global coordinates.*
- OAPIFUNC double [oapiGetPlanetCurrentRotation](#) (**OBJHANDLE** hPlanet)  
*Returns the current rotation angle of the planet around its axis.*
- OAPIFUNC bool [oapiPlanetHasAtmosphere](#) (**OBJHANDLE** hPlanet)  
*Test for existence of planetary atmosphere.*
- OAPIFUNC void [oapiGetPlanetAtmParams](#) (**OBJHANDLE** hPlanet, double rad, **ATMPARAM** \*prm)  
*Returns atmospheric parameters as a function of distance from the planet centre.*
- OAPIFUNC void [oapiGetPlanetAtmParams](#) (**OBJHANDLE** hPlanet, double alt, double lng, double lat, **ATMPARAM** \*prm)  
*Returns atmospheric parameters of a planet as a function of altitude and geographic position.*
- OAPIFUNC const **ATMCONST** \* [oapiGetPlanetAtmConstants](#) (**OBJHANDLE** hPlanet)  
*Returns atmospheric constants for a planet.*
- OAPIFUNC **VECTOR3** [oapiGetGroundVector](#) (**OBJHANDLE** hPlanet, double lng, double lat, int frame=2)  
*Returns the velocity vector of a surface point.*
- OAPIFUNC **VECTOR3** [oapiGetWindVector](#) (**OBJHANDLE** hPlanet, double lng, double lat, double alt, int frame=0)  
*Returns the wind velocity at a given position in a planet's atmosphere.*
- OAPIFUNC **DWORD** [oapiGetPlanetJCoeffCount](#) (**OBJHANDLE** hPlanet)  
*Returns the number of perturbation coefficients defined for a planet.*
- OAPIFUNC double [oapiGetPlanetJCoeff](#) (**OBJHANDLE** hPlanet, **DWORD** n)  
*Returns a perturbation coefficient for the calculation of a planet's gravitational potential.*
- OAPIFUNC **OBJHANDLE** [oapiGetBasePlanet](#) (**OBJHANDLE** hBase)  
*Returns a handle for the planet/moon the given base is located on.*
- OAPIFUNC void [oapiGetBaseEquPos](#) (**OBJHANDLE** hBase, double \*lng, double \*lat, double \*rad=0)  
*Returns the equatorial coordinates (longitude, latitude and radius) of the location of a surface base.*
- OAPIFUNC **DWORD** [oapiGetBasePadCount](#) (**OBJHANDLE** hBase)  
*Returns the number of VTOL landing pads owned by the base.*
- OAPIFUNC bool [oapiGetBasePadEquPos](#) (**OBJHANDLE** hBase, **DWORD** pad, double \*lng, double \*lat, double \*rad=0)

*Returns the equatorial coordinates (longitude, latitude and radius) of the location of a VTOL landing pad.*

- OAPIFUNC bool `oapiGetBasePadStatus (OBJHANDLE hBase, DWORD pad, int *status)`  
*Returns the status of a VTOL landing pad (free, occupied or cleared).*
- OAPIFUNC NAVHANDLE `oapiGetBasePadNav (OBJHANDLE hBase, DWORD pad)`  
*Returns a handle to the ILS transmitter of a VTOL landing pad, if available.*
- OAPIFUNC double `oapiGetSimTime ()`  
*Retrieve simulation time (in seconds) since simulation start.*
- OAPIFUNC double `oapiGetSimStep ()`  
*Retrieve length of last simulation time step (from previous to current frame) in seconds.*
- OAPIFUNC double `oapiGetSysTime ()`  
*Retrieve system (real) time since simulation start.*
- OAPIFUNC double `oapiGetSysStep ()`  
*Retrieve length of last system time step in seconds.*
- OAPIFUNC double `oapiGetSimMJD ()`  
*Retrieve absolute time measure (Modified Julian Date) for current simulation state.*
- OAPIFUNC double `oapiGetSysMJD ()`  
*Retrieve the current computer system time in Modified Julian Date (MJD) format.*
- OAPIFUNC bool `oapiSetSimMJD (double mjd, int pmode=0)`  
*Set the current simulation time. The simulation session performs a jump to the new time.*
- OAPIFUNC double `oapiTime2MJD (double simt)`  
*Convert a simulation up time value into a Modified Julian Date.*
- OAPIFUNC double `oapiGetTimeAcceleration ()`  
*Returns simulation time acceleration factor.*
- OAPIFUNC void `oapiSetTimeAcceleration (double warp)`  
*Set the simulation time acceleration factor.*
- OAPIFUNC double `oapiGetFrameRate ()`  
*Returns current simulation frame rate (frames/sec).*
- OAPIFUNC bool `oapiGetPause ()`  
*Returns the current simulation pause state.*
- OAPIFUNC void `oapiSetPause (bool pause)`  
*Sets the simulation pause state.*
- OAPIFUNC void `oapiGetNavPos (NAVHANDLE hNav, VECTOR3 *gpos)`  
*Returns the current position of a NAV transmitter (in global coordinates, i.e. heliocentric ecliptic).*

- OAPIFUNC DWORD `oapiGetNavChannel` (`NAVHANDLE hNav`)  
*Returns the channel number of a NAV transmitter.*
- OAPIFUNC float `oapiGetNavFreq` (`NAVHANDLE hNav`)  
*Returns the frequency of a NAV transmitter.*
- OAPIFUNC double `oapiGetNavSignal` (`NAVHANDLE hNav, const VECTOR3 &gpos`)  
*Returns the signal strength of a transmitter at a given position.*
- OAPIFUNC float `oapiGetNavRange` (`NAVHANDLE hNav`)  
*Returns the range of a NAV transmitter.*
- OAPIFUNC DWORD `oapiGetNavType` (`NAVHANDLE hNav`)  
*Returns the type id of a NAV transmitter.*
- OAPIFUNC int `oapiGetNavControllerData` (`NAVHANDLE hNav, NAVDATA *data`)  
*Returns information about a NAV transmitter.*
- OAPIFUNC int `oapiGetNavDescr` (`NAVHANDLE hNav, char *descr, int maxlen`)  
*Returns a descriptive string for a NAV transmitter.*
- OAPIFUNC bool `oapiNavInRange` (`NAVHANDLE hNav, const VECTOR3 &gpos`)  
*Determines whether a given global coordinate is within the range of a NAV transmitter.*
- OAPIFUNC `INTERPRETERHANDLE oapiCreateInterpreter()`  
*Returns a handle to a new interpreter instance.*
- OAPIFUNC int `oapiDeleteInterpreter` (`INTERPRETERHANDLE hInterp`)  
*Delete an interpreter instance.*
- OAPIFUNC bool `oapiExecScriptCmd` (`INTERPRETERHANDLE hInterp, const char *cmd`)  
*Executes a script command in an interpreter instance.*
- OAPIFUNC bool `oapiAsyncScriptCmd` (`INTERPRETERHANDLE hInterp, const char *cmd`)  
*Passes a command to an interpreter instance for execution.*
- OAPIFUNC `lua_State * oapiGetLua` (`INTERPRETERHANDLE hInterp`)
- OAPIFUNC `VISHANDLE * oapiGetObjectVisualPtr` (`OBJHANDLE hObject`)  
*Returns a pointer storing the objects visual handle.*
- OAPIFUNC `MESHHANDLE oapiLoadMesh` (`const char *fname`)  
*Loads a mesh from file and returns a handle to it.*
- OAPIFUNC `const MESHHANDLE oapiLoadMeshGlobal` (`const char *fname`)  
*Retrieves a mesh handle from the global mesh manager.*
- OAPIFUNC `const MESHHANDLE oapiLoadMeshGlobal` (`const char *fname, LoadMeshClbkFunc fClbk`)  
*Retrieves a mesh handle from the global mesh manager.*

- OAPIFUNC MESHHANDLE oapiCreateMesh (DWORD ngrp, MESHGROUP \*grp)  
*Creates a new mesh from a list of mesh group definitions.*
- OAPIFUNC void oapiDeleteMesh (MESHHANDLE hMesh)  
*Removes a mesh from memory.*
- OAPIFUNC DWORD oapiMeshGroupCount (MESHHANDLE hMesh)  
*Returns the number of mesh groups defined in a mesh.*
- OAPIFUNC MESHGROUP \* oapiMeshGroup (MESHHANDLE hMesh, DWORD idx)  
*Returns a pointer to the group specification of a mesh group.*
- OAPIFUNC MESHGROUP \* oapiMeshGroup (DEVMESHHANDLE hMesh, DWORD idx)
- OAPIFUNC MESHGROUPEX \* oapiMeshGroupEx (MESHHANDLE hMesh, DWORD idx)
- OAPIFUNC DWORD oapiAddMeshGroup (MESHHANDLE hMesh, MESHGROUP \*grp)
- OAPIFUNC bool oapiAddMeshGroupBlock (MESHHANDLE hMesh, DWORD grpidx, const NTVERTEX \*vtx, DWORD nvtx, const WORD \*idx, DWORD nidx)
- OAPIFUNC int oapiEditMeshGroup (MESHHANDLE hMesh, DWORD grpidx, GROUPEDITSPEC \*ges)  
*Modify mesh group data.*
- OAPIFUNC int oapiEditMeshGroup (DEVMESHHANDLE hMesh, DWORD grpidx, GROUPEDITSPEC \*ges)
- OAPIFUNC DWORD oapiMeshTextureCount (MESHHANDLE hMesh)  
*Returns the number of textures associated with a mesh.*
- OAPIFUNC SURFHANDLE oapiGetTextureHandle (MESHHANDLE hMesh, DWORD texidx)  
*Retrieve a surface handle for a mesh texture.*
- OAPIFUNC SURFHANDLE oapiLoadTexture (const char \*fname, bool dynamic=false)  
*Load a texture from a file.*
- OAPIFUNC void oapiReleaseTexture (SURFHANDLE hTex)  
*Release a texture.*
- OAPIFUNC bool oapiSetTexture (MESHHANDLE hMesh, DWORD texidx, SURFHANDLE tex)  
*Replace a mesh texture.*
- OAPIFUNC bool oapiSetTexture (DEVMESHHANDLE hMesh, DWORD texidx, SURFHANDLE tex)
- OAPIFUNC DWORD oapiMeshMaterialCount (MESHHANDLE hMesh)  
*Returns the number of materials defined in the mesh.*
- OAPIFUNC MATERIAL \* oapiMeshMaterial (MESHHANDLE hMesh, DWORD idx)  
*Returns a pointer to a material specification in the material list of the mesh.*
- OAPIFUNC DWORD oapiAddMaterial (MESHHANDLE hMesh, MATERIAL \*mat)  
*Add a material definition to a mesh.*
- OAPIFUNC bool oapiDeleteMaterial (MESHHANDLE hMesh, DWORD idx)

*Delete a material definition from the mesh.*

- OAPIFUNC int [oapiSetMaterial](#) (**DEVMESHHANDLE** hMesh, DWORD matidx, const **MATERIAL** \*mat)

*Reset the properties of a mesh material.*

- OAPIFUNC bool [oapiSetMeshProperty](#) (**MESHHANDLE** hMesh, DWORD property, DWORD value)

*Set custom properties for a mesh.*

- OAPIFUNC bool [oapiSetMeshProperty](#) (**DEVMESHHANDLE** hMesh, DWORD property, DWORD value)

*Set custom properties for a device-specific mesh.*

- OAPIFUNC void [oapiParticleSetLevelRef](#) (**PSTREAM\_HANDLE** ph, double \*lvl)

*Reset the reference pointer used by the particle stream to calculate the intensity (opacity) of the generated particles.*

- OAPIFUNC bool [oapiSetHUDMode](#) (int mode)

*Set HUD (head up display) mode.*

- OAPIFUNC bool [oapiSetHUDMode](#) (int mode, const **HUDPARAM** \*prm)

*Set HUD (head up display) mode with mode-specific parameters.*

- OAPIFUNC int [oapiGetHUDMode](#) ()

*Query current HUD (head up display) mode.*

- OAPIFUNC int [oapiGetHUDMode](#) (**HUDPARAM** \*prm)

*Query current HUD mode and mode parameters.*

- OAPIFUNC void [oapiToggleHudColour](#) ()

*Switch the HUD display to a different colour.*

- OAPIFUNC void [oapiIncHUDIntensity](#) ()

*Increase the brightness of the HUD display.*

- OAPIFUNC void [oapiDecHUDIntensity](#) ()

*Decrease the brightness of the HUD display.*

- OAPIFUNC void [oapiRenderHUD](#) (**MESHHANDLE** hMesh, **SURFHANDLE** \*hTex)

*Render custom HUD elements.*

- OAPIFUNC void [oapiOpenMFD](#) (int mode, int mfd)

*Set an **MFD** (multifunctional display) to a specific mode.*

- OAPIFUNC void [oapiToggleMFD\\_on](#) (int mfd)

*Switches an **MFD** on or off.*

- OAPIFUNC int [oapiGetMFDMode](#) (int mfd)

*Get the current mode of the specified **MFD**.*

- OAPIFUNC int **oapiBroadcastMFDMessage** (int mode, int msg, void \*data)  
• OAPIFUNC int **oapiSendMFDKey** (int mfd, DWORD key)  
*Sends a keystroke to an MFD.*
- OAPIFUNC void **oapiRefreshMFDButtons** (int mfd, **OBJHANDLE** hVessel=0)  
*Sends a clbkMFDMode call to the current focus vessel to allow it to dynamically update its button labels.*
- OAPIFUNC bool **oapiProcessMFDButton** (int mfd, int bt, int event)  
*Requests a default action as a result of a MFD button event.*
- OAPIFUNC const char \* **oapiMFDButtonLabel** (int mfd, int bt)  
*Retrieves a default label for an MFD button.*
- OAPIFUNC void **oapiRegisterMFD** (int mfd, const MFDSPEC &spec)  
*Registers an MFD position for a custom panel.*
- OAPIFUNC void **oapiRegisterMFD** (int mfd, const EXTMFDSPEC \*spec)  
*Registers an MFD position for a custom panel or virtual cockpit. This version has an extended parameter list.*
- OAPIFUNC void **oapiRegisterExternMFD** (**ExternMFD** \*emfd, const MFDSPEC &spec)  
• OAPIFUNC bool **oapiUnregisterExternMFD** (**ExternMFD** \*emfd)  
• OAPIFUNC void **oapiRegisterPanelBackground** (HBITMAP hBmp, DWORD flag=PANEL\_ATTACH\_BOTTOM|PANEL\_MOVEOUT\_BOTTOM, DWORD ck=(DWORD)-1)  
*Register the background bitmap for a custom panel.*
- OAPIFUNC void **oapiRegisterPanelArea** (int id, const RECT &pos, int draw\_event=PANEL\_REDRAW\_NEVER, int mouse\_event=PANEL\_MOUSE\_IGNORE, int bkmode=PANEL\_MAP\_NONE)  
*Defines a rectangular area within a panel to receive mouse or redraw notifications.*
- OAPIFUNC void **oapiSetPanelNeighbours** (int left, int right, int top, int bottom)  
*Defines the neighbour panels of the current panels. These are the panels the user can switch to via Ctrl-Arrow keys.*
- OAPIFUNC void **oapiTriggerPanelRedrawArea** (int panel\_id, int area\_id)  
*Triggers a redraw notification for a panel area.*
- OAPIFUNC void **oapiTriggerRedrawArea** (int panel\_id, int vc\_id, int area\_id)  
*Triggers a redraw notification to either a 2D panel or a virtual cockpit.*
- OAPIFUNC bool **oapiBltPanelAreaBackground** (int area\_id, **SURFHANDLE** surf)  
*Copies the stored background of a panel area into the provided surface.*
- OAPIFUNC void **oapiSetDefNavDisplay** (int mode)  
*Defines how the navigation mode buttons will be displayed in a default cockpit view.*
- OAPIFUNC void **oapiSetDefRCSDisplay** (int mode)  
*Enable or disable the display of the reaction control system indicators/controls in default cockpit view.*

- OAPIFUNC int [oapiSwitchPanel](#) (int direction)  
*Switch to a neighbour instrument panel in 2-D panel cockpit mode.*
- OAPIFUNC int [oapiSetPanel](#) (int panel\_id)  
*Switch to a different instrument panel in 2-D panel cockpit mode.*
- OAPIFUNC void [oapiSetPanelBlink](#) ([VECTOR3](#) v[4])
- OAPIFUNC [oapi::Sketchpad](#) \* [oapiGetSketchpad](#) ([SURFHANDLE](#) surf)  
*Obtain a drawing context for a surface.*
- OAPIFUNC void [oapiReleaseSketchpad](#) ([oapi::Sketchpad](#) \*skp)  
*Release a drawing device context instance.*
- OAPIFUNC [oapi::Font](#) \* [oapiCreateFont](#) (int height, bool prop, char \*face, FontStyle style=FONT\_NORMAL)  
*Creates a font resource for drawing text into surfaces.*
- OAPIFUNC [oapi::Font](#) \* [oapiCreateFont](#) (int height, bool prop, const char \*face, FontStyle style, int orientation)  
*Creates a font resource for drawing text into surfaces.*
- OAPIFUNC void [oapiReleaseFont](#) ([oapi::Font](#) \*font)  
*Release a font resource.*
- OAPIFUNC [oapi::Pen](#) \* [oapiCreatePen](#) (int style, int width, DWORD col)  
*Creates a pen resource for drawing lines and shape outlines.*
- OAPIFUNC void [oapiReleasePen](#) ([oapi::Pen](#) \*pen)  
*Release a pen resource.*
- OAPIFUNC [oapi::Brush](#) \* [oapiCreateBrush](#) (DWORD col)  
*Creates a brush resource for filling shapes.*
- OAPIFUNC void [oapiReleaseBrush](#) ([oapi::Brush](#) \*brush)  
*Release a brush resource.*
- OAPIFUNC HDC [oapiGetDC](#) ([SURFHANDLE](#) surf)  
*Obtain a Windows device context handle (HDC) for a surface.*
- OAPIFUNC void [oapiReleaseDC](#) ([SURFHANDLE](#) surf, HDC hDC)  
*Release a GDI drawing device context handle.*
- OAPIFUNC [SURFHANDLE](#) [oapiCreateSurface](#) (int width, int height)  
*Create a surface of the specified dimensions.*
- OAPIFUNC [SURFHANDLE](#) [oapiCreateSurface](#) (HBITMAP hBmp, bool release\_bmp=true)  
*Create a surface from a bitmap. Bitmap surfaces are typically used for blitting operations during instrument panel redraws.*

- OAPIFUNC SURFHANDLE oapiCreateTextureSurface (int width, int height)  
*Create a surface that can be used as a texture for a 3-D object.*
- OAPIFUNC void oapiDestroySurface (SURFHANDLE surf)  
*Destroy a surface previously created with oapiCreateSurface.*
- OAPIFUNC void oapiClearSurface (SURFHANDLE surf, DWORD col=0)
- OAPIFUNC void oapiSetSurfaceColourKey (SURFHANDLE surf, DWORD ck)  
*Define a colour key for a surface to allow transparent blitting.*
- OAPIFUNC void oapiClearSurfaceColourKey (SURFHANDLE surf)  
*Clear a previously defined colour key.*
- OAPIFUNC void oapiBlt (SURFHANDLE tgt, SURFHANDLE src, int tgtx, int tgyt, int srcx, int srcy, int w, int h, DWORD ck=SURF\_NO\_CK)  
*Copy a rectangular area from one surface to another.*
- OAPIFUNC void oapiBlt (SURFHANDLE tgt, SURFHANDLE src, RECT \*tgtr, RECT \*srcr, DWORD ck=SURF\_NO\_CK, DWORD rotate=SURF\_NO\_ROTATION)  
*Copy a scaled rectangular area from one surface to another.*
- OAPIFUNC int oapiBeginBltGroup (SURFHANDLE tgt)  
*Begin a block of blitting operations to the same target surface.*
- OAPIFUNC int oapiEndBltGroup ()  
*End a block of blitting operations to the same target surface.*
- OAPIFUNC void oapiColourFill (SURFHANDLE tgt, DWORD fillcolor, int tgtx=0, int tgyt=0, int w=0, int h=0)  
*Fill an area of the target surface with a uniform colour.*
- OAPIFUNC int oapiRegisterMFDMode (MFDMODESPECEX &spec)  
*Register a custom MFD mode.*
- OAPIFUNC bool oapiUnregisterMFDMode (int mode)  
*Unregister a previously registered custom MFD mode.*
- OAPIFUNC void oapiDisableMFDMode (int mode)  
*Disable an MFD mode.*
- OAPIFUNC int oapiGetMFDModeSpecEx (char \*name, MFDMODESPECEX \*\*spec=0)  
*Returns the mode identifier and spec for an MFD mode defined by its name.*
- OAPIFUNC void oapiVCRegisterMFD (int mfd, const VCMFDSPEC \*spec)  
*Define a render target for rendering an MFD display in a virtual cockpit.*
- OAPIFUNC void oapiVCRegisterArea (int id, const RECT &tgtrect, int draw\_event, int mouse\_event, int bkmode, SURFHANDLE tgt)  
*Define an active area in a virtual cockpit. Active areas can be repainted. This function is similar to oapiRegisterPanelArea.*

- OAPIFUNC void [oapiVCRegisterArea](#) (int id, int draw\_event, int mouse\_event)  
*Define an active area in a virtual cockpit. This version is used when no dynamic texture update is required during redraw events.*
- OAPIFUNC void [oapiVCSetAreaClickmode\\_Spherical](#) (int id, const [VECTOR3](#) &cnt, double rad)  
*Associate a spherical region in the virtual cockpit with a registered area to receive mouse events.*
- OAPIFUNC void [oapiVCSetAreaClickmode\\_Quadrilateral](#) (int id, const [VECTOR3](#) &p1, const [VECTOR3](#) &p2, const [VECTOR3](#) &p3, const [VECTOR3](#) &p4)  
*Associate a quadrilateral region in the virtual cockpit with a registered area to receive mouse events.*
- OAPIFUNC void [oapiVCSetNeighbours](#) (int left, int right, int top, int bottom)  
*Defines the neighbouring virtual cockpit camera positions in relation to the current position. The user can switch to neighbour positions with Ctrl-Arrow keys.*
- OAPIFUNC void [oapiVCTriggerRedrawArea](#) (int vc\_id, int area\_id)  
*Triggers a redraw notification for a virtual cockpit area.*
- OAPIFUNC void [oapiVCRegisterHUD](#) (const [VCHUDSPEC](#) \*spec)  
*Define a render target for the head-up display (HUD) in a virtual cockpit.*
- OAPIFUNC [LAUNCHPADITEM\\_HANDLE](#) [oapiRegisterLaunchpadItem](#) ([LaunchpadItem](#) \*item, [LAUNCHPADITEM\\_HANDLE](#) parent=0)  
*Register a new item in the parameter list of the "Extra" tab of the Orbiter Launchpad dialog.*
- OAPIFUNC bool [oapiUnregisterLaunchpadItem](#) ([LaunchpadItem](#) \*item)  
*Unregister a previously registered entry in the "Extra" tab of the Orbiter Launchpad dialog.*
- OAPIFUNC [LAUNCHPADITEM\\_HANDLE](#) [oapiFindLaunchpadItem](#) (const char \*name=0, [LAUNCHPADITEM\\_HANDLE](#) parent=0)  
*Returns a handle for an existing entry in the Extra parameter list.*
- OAPIFUNC DWORD [oapiRegisterCustomCmd](#) (char \*label, char \*desc, CustomFunc func, void \*context)  
*Register a custom function. Custom functions can be accessed in Orbiter by pressing Ctrl-F4. A common use for custom functions is opening plugin dialog boxes.*
- OAPIFUNC bool [oapiUnregisterCustomCmd](#) (int cmdId)  
*Unregister a previously defined custom function.*
- OAPIFUNC HWND [oapiOpenDialog](#) (HINSTANCE hDLLInst, int resourceId, DLGPROC msgProc, void \*context=0)  
*Open a dialog box defined as a Windows resource.*
- OAPIFUNC HWND [oapiOpenDialogEx](#) (HINSTANCE hDLLInst, int resourceId, DLGPROC msgProc, DWORD flag=0, void \*context=0)  
*Open a dialog box defined as a Windows resource. This version provides additional functionality compared to [oapiOpenDialog\(\)](#).*
- OAPIFUNC HWND [oapiFindDialog](#) (HINSTANCE hDLLInst, int resourceId)

*Returns the window handle of an open dialog box, or NULL if the specified dialog box is not open.*

- OAPIFUNC void [oapiCloseDialog](#) (HWND hDlg)  
*Close a dialog box.*
- OAPIFUNC void \* [oapiGetDialogContext](#) (HWND hDlg)  
*Retrieves the context pointer of a dialog box which has been defined during the call to [oapiOpenDialog\(\)](#).*
- OAPIFUNC bool [oapiRegisterWindow](#) (HINSTANCE hDLLInst, HWND hWnd, DWORD flag=0)
- OAPIFUNC bool [oapiAddTitleButton](#) (DWORD msgid, HBITMAP hBmp, DWORD flag)  
*Adds a custom button in the title bar of a dialog box.*
- OAPIFUNC DWORD [oapiGetTitleButtonState](#) (HWND hDlg, DWORD msgid)
- OAPIFUNC bool [oapiSetTitleButtonState](#) (HWND hDlg, DWORD msgid, DWORD state)
- OAPIFUNC BOOL [oapiDefDialogProc](#) (HWND hDlg, UINT uMsg, WPARAM wParam, LPARAM lParam)  
*Default Orbiter dialog message handler.*
- OAPIFUNC bool [oapiOpenHelp](#) (HELPCONTEXT \*hcontext)  
*Opens the ingame help window on the specified help page.*
- OAPIFUNC bool [oapiOpenLaunchpadHelp](#) (HELPCONTEXT \*hcontext)  
*Opens a help window outside a simulation session, i.e. when the Launchpad dialog is displayed.*
- OAPIFUNC FILEHANDLE [oapiOpenFile](#) (const char \*fname, FileAccessMode mode, PathRoot root=ROOT)  
*Open a file for reading or writing.*
- OAPIFUNC void [oapiCloseFile](#) (FILEHANDLE file, FileAccessMode mode)  
*Close a file after reading or writing.*
- OAPIFUNC bool [oapiSaveScenario](#) (const char \*fname, const char \*desc)  
*Writes the current simulation state to a scenario file.*
- OAPIFUNC void [oapiWriteLine](#) (FILEHANDLE file, char \*line)  
*Writes a line to a file.*
- OAPIFUNC void [oapiWriteLog](#) (char \*line)  
*Writes a line to the Orbiter log file (orbiter.log) in the main orbiter directory.*
- OAPIFUNC void [oapiWriteLogV](#) (const char \*format,...)  
*Writes a formatted string with variable number of arguments to orbiter.log.*
- OAPIFUNC void [oapiWriteScenario\\_string](#) (FILEHANDLE scn, char \*item, char \*string)  
*Writes a string-valued item to a scenario file.*
- OAPIFUNC void [oapiWriteScenario\\_int](#) (FILEHANDLE scn, char \*item, int i)  
*Writes an integer-valued item to a scenario file.*

- OAPIFUNC void `oapiWriteScenario_float` (`FILEHANDLE` scn, `char *item`, `double d`)  
*Writes a floating point-valued item to a scenario file.*
- OAPIFUNC void `oapiWriteScenario_vec` (`FILEHANDLE` scn, `char *item`, `const VECTOR3 &vec`)  
*Writes a vector-valued item to a scenario file.*
- OAPIFUNC bool `oapiReadScenario_nextline` (`FILEHANDLE` scn, `char *&line`)  
*Reads an item from a scenario file.*
- OAPIFUNC bool `oapiReadItem_string` (`FILEHANDLE` f, `char *item`, `char *string`)  
*Read the value of a tag from a configuration file.*
- OAPIFUNC bool `oapiReadItem_float` (`FILEHANDLE` f, `char *item`, `double &d`)  
*Read the value of a tag from a configuration file.*
- OAPIFUNC bool `oapiReadItem_int` (`FILEHANDLE` f, `char *item`, `int &i`)  
*Read the value of a tag from a configuration file.*
- OAPIFUNC bool `oapiReadItem_bool` (`FILEHANDLE` f, `char *item`, `bool &b`)  
*Read the value of a tag from a configuration file.*
- OAPIFUNC bool `oapiReadItem_vec` (`FILEHANDLE` f, `char *item`, `VECTOR3 &vec`)  
*Read the value of a tag from a configuration file.*
- OAPIFUNC void `oapiWriteItem_string` (`FILEHANDLE` f, `char *item`, `char *string`)  
*Write a tag and its value to a configuration file.*
- OAPIFUNC void `oapiWriteItem_float` (`FILEHANDLE` f, `char *item`, `double d`)  
*Write a tag and its value to a configuration file.*
- OAPIFUNC void `oapiWriteItem_int` (`FILEHANDLE` f, `char *item`, `int i`)  
*Write a tag and its value to a configuration file.*
- OAPIFUNC void `oapiWriteItem_bool` (`FILEHANDLE` f, `char *item`, `bool b`)  
*Write a tag and its value to a configuration file.*
- OAPIFUNC void `oapiWriteItem_vec` (`FILEHANDLE` f, `char *item`, `const VECTOR3 &vec`)  
*Write a tag and its value to a configuration file.*
- OAPIFUNC double `oapiRand` ()  
*Returns uniformly distributed pseudo-random number in the range [0..1].*
- OAPIFUNC `DWORD oapiGetColour` (`DWORD red`, `DWORD green`, `DWORD blue`)  
*Returns a colour value adapted to the current screen colour depth for given red, green and blue components.*
- OAPIFUNC void `oapiOpenInputBox` (`char *title`, `bool(*Cblk)(void *, char *, void *)`, `char *buf=0`, `int vislen=20`, `void *usrdata=0`)  
*Opens a modal input box requesting a string from the user.*

- OAPIFUNC void **oapiOpenInputBoxEx** (const char \*title, bool(\*Clbk\_enter)(void \*, char \*, void \*), bool(\*Clbk\_cancel)(void \*, char \*, void \*), char \*buf=0, int vislen=20, void \*usrdta=0, DWORD flags=0)
  - OAPIFUNC NOTEHANDLE **oapiCreateAnnotation** (bool exclusive, double size, const **VECTOR3** &col)
    - Creates an annotation handle for displaying onscreen text during a simulation.*
  - OAPIFUNC bool **oapiDelAnnotation** (NOTEHANDLE hNote)
    - Deletes an annotation handle.*
  - OAPIFUNC void **oapiAnnotationSetPos** (NOTEHANDLE hNote, double x1, double y1, double x2, double y2)
    - Resets the bounding box of the annotation display area.*
  - OAPIFUNC void **oapiAnnotationSetSize** (NOTEHANDLE hNote, double size)
    - Resets the font size of the annotation text.*
  - OAPIFUNC void **oapiAnnotationSetColour** (NOTEHANDLE hNote, const **VECTOR3** &col)
    - Resets the font colour of the annotation text.*
  - OAPIFUNC void **oapiAnnotationSetText** (NOTEHANDLE hNote, char \*note)
    - Writes a new annotation to screen, or overwrites the previous text.*
- OAPIFUNC OBJHANDLE **oapiGetStationByName** (char \*name)
  - OAPIFUNC OBJHANDLE **oapiGetStationByIndex** (int index)
  - OAPIFUNC DWORD **oapiGetStationCount** ()
  - OAPIFUNC BOOL **oapiGetAirspeedVector** (OBJHANDLE hVessel, **VECTOR3** \*speedvec)
    - Returns a vessel's airspeed vector w.r.t. the closest planet or moon in the local horizon's frame of reference.*
  - OAPIFUNC BOOL **oapiGetShipAirspeedVector** (OBJHANDLE hVessel, **VECTOR3** \*speedvec)
    - Returns a vessel's airspeed vector w.r.t. the closest planet or moon in the vessel's local frame of reference.*
  - OAPIFUNC BOOL **oapiGetFocusAirspeed** (double \*airspeed)
    - Returns the current focus vessel's airspeed w.r.t. the closest planet or moon.*
  - OAPIFUNC BOOL **oapiGetFocusAirspeedVector** (**VECTOR3** \*speedvec)
    - Returns the current focus vessel's airspeed vector w.r.t. the closest planet or moon in the local horizon's frame of reference.*
  - OAPIFUNC BOOL **oapiGetFocusShipAirspeedVector** (**VECTOR3** \*speedvec)
    - Returns the current focus vessel's airspeed vector w.r.t. closest planet or moon in the vessel's local frame of reference.*
  - OAPIFUNC void **oapiGetAtmPressureDensity** (OBJHANDLE hVessel, double \*pressure, double \*density)
    - Returns the atmospheric pressure and density caused by a planetary atmosphere at the current vessel position.*
  - OAPIFUNC void **oapiGetFocusAtmPressureDensity** (double \*pressure, double \*density)
    - Returns the atmospheric pressure and density caused by a planetary atmosphere at the current focus vessel's position.*

- OAPIFUNC bool **oapiAcceptDelayedKey** (char key, double interval)  
**oapiRegisterMFDMode** (MFDMODESPEC &spec)  
*Register a custom MFD mode.*
- OAPIFUNC int **oapiGetMFDModeSpec** (char \*name, MFDMODESPEC \*\*spec=0)  
*Returns the mode identifier and spec for an MFD mode defined by its name.*
- **VECTOR3 \_V** (double x, double y, double z)  
*Vector composition.*
- void **veccpy** (VECTOR3 &a, const VECTOR3 &b)  
*Vector copy.*
- **VECTOR3 operator+** (const VECTOR3 &a, const VECTOR3 &b)  
*Vector addition.*
- **VECTOR3 operator-** (const VECTOR3 &a, const VECTOR3 &b)  
*Vector subtraction.*
- **VECTOR3 operator\*** (const VECTOR3 &a, const double f)  
*Multiplication of vector with scalar.*
- **VECTOR3 operator/** (const VECTOR3 &a, const double f)  
*Division of vector by a scalar.*
- **VECTOR3 & operator+=** (VECTOR3 &a, const VECTOR3 &b)  
*Vector addition-assignment  $a += b$ .*
- **VECTOR3 & operator-=** (VECTOR3 &a, const VECTOR3 &b)  
*Vector subtraction-assignment  $a -= b$ .*
- **VECTOR3 & operator\*=** (VECTOR3 &a, const double f)  
*Vector-scalar multiplication-assignment  $a *= f$ .*
- **VECTOR3 & operator/=** (VECTOR3 &a, const double f)  
*Vector-scalar division-assignment  $a /= f$ .*
- **VECTOR3 operator-** (const VECTOR3 &a)  
*Vector unary minus  $-a$ .*
- double **dotp** (const VECTOR3 &a, const VECTOR3 &b)  
*Scalar (inner, dot) product of two vectors.*
- **VECTOR3 crossp** (const VECTOR3 &a, const VECTOR3 &b)  
*Vector (cross) product of two vectors.*
- double **length** (const VECTOR3 &a)  
*Length (L2-norm) of a vector.*

- double **dist** (const **VECTOR3** &a, const **VECTOR3** &b)  
*Distance between two points.*
- void **normalise** (**VECTOR3** &a)  
*Normalise a vector.*
- **VECTOR3 unit** (const **VECTOR3** &a)  
*Returns normalised vector.*
- **MATRIX3 \_M** (double m11, double m12, double m13, double m21, double m22, double m23, double m31, double m32, double m33)  
*Matrix composition.*
- **MATRIX3 identity** ()  
*Returns the identity matrix.*
- **MATRIX3 outerp** (const **VECTOR3** &a, const **VECTOR3** &b)  
*Outer product of two vectors.*
- **MATRIX3 operator+** (const **MATRIX3** &A, double s)  
*Sum of matrix and scalar.*
- **MATRIX3 operator-** (const **MATRIX3** &A, double s)  
*Difference of matrix and scalar.*
- **MATRIX3 operator\*** (const **MATRIX3** &A, double s)  
*Product of matrix and scalar.*
- **MATRIX3 operator/** (const **MATRIX3** &A, double s)  
*Quotient of matrix and scalar.*
- **MATRIX3 & operator\*=(** **MATRIX3** &A, double s)  
*Matrix-scalar product-assignment  $A *= s$ .*
- **MATRIX3 & operator/=(** **MATRIX3** &A, double s)  
*Matrix-scalar division-assignment  $A /= s$ .*
- **VECTOR3 mul** (const **MATRIX3** &A, const **VECTOR3** &b)  
*Matrix-vector multiplication.*
- **VECTOR3 tmul** (const **MATRIX3** &A, const **VECTOR3** &b)  
*Matrix transpose-vector multiplication.*
- **MATRIX3 mul** (const **MATRIX3** &A, const **MATRIX3** &B)  
*Matrix-matrix multiplication.*
- **RECT \_R** (int left, int top, int right, int bottom)
- **VECTOR3 POINTERTOREF** (**VECTOR3** \*p)

## Variables

- const double **PI** = 3.14159265358979323846  
*pi*
- const double **PI05** = 1.57079632679489661923  
*pi/2*
- const double **PI2** = 6.28318530717958647693  
*pi\*2*
- const double **RAD** = **PI**/180.0  
*factor to map degrees to radians*
- const double **DEG** = 180.0/**PI**  
*factor to map radians to degrees*
- const double **C0** = 299792458.0  
*speed of light in vacuum [m/s]*
- const double **TAUA** = 499.004783806  
*light time for 1 AU [s]*
- const double **AU** = **C0**\***TAUA**  
*astronomical unit (mean geocentric distance of the sun) [m]*
- const double **GGRAV** = 6.67259e-11  
*gravitational constant [m^3 kg^-1 s^-2]*
- const double **G** = 9.81  
*gravitational acceleration [m/s^2] at Earth mean radius*
- const double **ATMP** = 101.4e3  
*atmospheric pressure [Pa] at Earth sea level*
- const double **ATMD** = 1.293  
*atmospheric density [kg/m^3] at Earth sea level*
- const DWORD **MAXTEX** = 1
- const UINT **ALLDOCKS** = (UINT)-1

### 17.4.2 Function Documentation

#### 17.4.2.1 double normangle (double *angle*) [inline]

Returns the input argument normalised to range -pi ... pi.

##### Parameters:

*angle* input angle [rad]

**Returns:**

normalised angle [rad]

**17.4.2.2 double posangle (double *angle*) [inline]**

Returns the input argument normalised to range 0 ... 2 pi.

**Parameters:**

*angle* input angle [rad]

**Returns:**

normalised angle [rad]

**17.5 Orbitersdk/include/VesselAPI.h File Reference****17.5.1 Detailed Description**

Contains the class interfaces for vessel objects ([VESSEL](#), [VESSEL2](#), [VESSEL3](#)).

**Classes**

- class [VESSEL](#)  
*Base class for objects of vessel type (spacecraft and similar).*
- class [VESSEL2](#)  
*Callback extensions to the [VESSEL](#) class.*
- class [VESSEL3](#)  
*Callback extensions to the [VESSEL](#) class.*
- class [VESSEL4](#)  
*Extensions to the [VESSEL](#) class.*

**Defines**

- #define **FRAME\_ECL** 0
- #define **FRAME\_EQU** 1

**18 Example Documentation****18.1 clbkLoadStateEx.cpp**

Example of an overloaded [VESSEL2::clbkLoadStateEx](#) method.

```

class MyVessel: public VESSEL2 {
public:
    ...
    clbkLoadStateEx (FILEHANDLE scn, void *status);
    ...
};

void MyVessel::clbkLoadStateEx (FILEHANDLE scn, void *status)
{
    char *line;
    int my_value;

    while (oapiReadScenario_nextline (scn, line)) {
        if (!strnicmp (line, "my_option", 9)) { // custom item
            sscanf (line+9, "%d", &my_value);
        } else if (...) { // more items
            ...
        } else { // anything not recognised is passed on to Orbiter
            ParseScenarioLineEx (line, vs);
        }
    }
}

```

## 18.2 clbkPreStep.cpp

Example of an overloaded [VESSEL2::clbkPreStep](#) method.

```

class MyVessel: public VESSEL2 {
public:
    ...
    clbkPreStep (double simt, double simdt, double mjd);
    ...
};

void MyVessel::clbkPreStep (double simt, double simdt, double mjd)
{
    double F = mass * dv/simdt;
    AddForce(_V(0,0,F), _V(0,0,0));
}

```

## 18.3 clbkSetStateEx.cpp

Example of an overloaded [VESSEL2::clbkSetStateEx](#) method.

```

class MyVessel: public VESSEL2 {
public:
    ...
    clbkSetStateEx (const void *status);
    ...
};

void MyVessel::clbkSetStateEx (const void *status)
{
    // specialised vessel initialisations
    // ...

    // default initialisation:
    DefSetStateEx (status);
}

```

## 19 VESSEL2.cpp

Example for constructing and destroying an overloaded **VESSEL2** (p. 503) instance during the instance initialisation of a vessel module.

```
1 class MyVessel: public VESSEL2 {
2 public:
3     MyVessel (OBJHANDLE hvessel, int flightmodel = 1);
4     ...
5 };
6
7 MyVessel::MyVessel (OBJHANDLE hvessel, int flightmodel)
8 : VESSEL2 (hvessel, flightmodel)
9 {
10     ...
11 }
12
13 DLLCLBK VESSEL2 *ovcInit (OBJHANDLE hvessel, int flightmodel)
14 {
15     return new MyVessel (hvessel, flightmodel);
16 }
17
18 DLLCLBK void ovcExit (VESSEL2 *vessel)
19 {
20     delete (MyVessel*)vessel;
21 }
```

# Index

~CELBODY2  
    CELBODY2, 222

~ExternMFD  
    ExternMFD, 231

~GraphicsClient  
    oapi::GraphicsClient, 243

\_M  
    vec, 33

\_V  
    vec, 34

Activate  
    LightEmitter, 288

ActivateNavmode  
    VESSEL, 394

Active  
    ExternMFD, 231

AddAnimationComponent  
    VESSEL, 469

AddBeacon  
    VESSEL, 494

AddExhaust  
    VESSEL, 486–488

AddExhaustStream  
    VESSEL, 491, 492

AddForce  
    VESSEL, 423

AddGraph  
    GraphMFD, 278

AddMesh  
    VESSEL, 460, 461

AddParticleStream  
    VESSEL, 491

AddPlot  
    GraphMFD, 278

AddPointLight  
    VESSEL, 496

AddReentryStream  
    VESSEL, 492

AddSpotLight  
    VESSEL, 496

Aerodynamic control surface types, 47

AIRCTRL\_AILERON  
    airctrltype, 47

AIRCTRL\_ELEVATOR  
    airctrltype, 47

AIRCTRL\_ELEVATORTRIM  
    airctrltype, 47

AIRCTRL\_FLAP  
    airctrltype, 47

AIRCTRL\_RUDDER  
    airctrltype, 47

airctrltype, 47

AIRCTRL\_RUDDERTRIM  
    airctrltype, 47

AIRCTRL\_AXIS\_AUTO  
    airctrlaxis, 48

AIRCTRL\_TYPE  
    airctrltype, 47

airctrlaxis  
    AIRCTRL\_AXIS\_AUTO, 48

airctrltype  
    AIRCTRL\_AILERON, 47  
    AIRCTRL\_ELEVATOR, 47  
    AIRCTRL\_ELEVATORTRIM, 47  
    AIRCTRL\_FLAP, 47  
    AIRCTRL\_RUDDER, 47  
    AIRCTRL\_RUDDERTRIM, 47  
    AIRCTRL\_TYPE, 47

Airfoil orientation, 46

AIRFOIL\_ORIENTATION  
    airfoilliftdir, 46

airfoilliftdir  
    AIRFOIL\_ORIENTATION, 46  
    LIFT\_HORIZONTAL, 47  
    LIFT\_VERTICAL, 47

ANIMATION, 206

Animation flags, 44

ANIMATIONCOMP, 207

Annotations  
    oapiAnnotationSetColour, 181  
    oapiAnnotationSetPos, 181  
    oapiAnnotationSetSize, 182  
    oapiAnnotationSetText, 182  
    oapiCreateAnnotation, 182  
    oapiDelAnnotation, 182

arot  
    VESSELSTATUS2, 536

ATMCONST, 207

ATMOSPHERE, 209  
    ATMOSPHERE, 210  
    clbkConstants, 210  
    clbkName, 210  
    clbkParams, 211  
    PRM\_ALT, 210  
    PRM\_AP, 210  
    PRM\_F, 210  
    PRM\_FBR, 210  
    PRM\_LAT, 210  
    PRM\_LNG, 210  
    PRM\_IN\_FLAG, 210

ATMOSPHERE::PRM\_IN, 211

ATMOSPHERE::PRM\_OUT, 212  
 ATMPARAM, 213  
 Attach  
     oapi::ParticleStream, 323, 324  
 AttachChild  
     VESSEL, 485  
 AttachmentCount  
     VESSEL, 484  
 BaseInterface  
     oapiGetBaseEquPos, 109  
     oapiGetBasePadCount, 110  
     oapiGetBasePadEquPos, 110  
     oapiGetBasePadNav, 110  
     oapiGetBasePadStatus, 111  
     oapiGetBasePlanet, 111  
 BASELINE  
     oapi::Sketchpad, 337  
 BEACONLIGHTSPEC, 213  
 bEphemeris  
     CELBODY, 217  
 Bit flags for blitting operations, 27  
 Bit flags for planetarium mode elements, 26  
 Bitflags for EXHAUSTSPEC flags field., 43  
 BK\_OPAQUE  
     oapi::Sketchpad, 338  
 BK\_TRANSPARENT  
     oapi::Sketchpad, 338  
 BkgMode  
     oapi::Sketchpad, 337  
 BLT\_TGTCOLORKEY  
     bltflag, 27  
 bltflag  
     BLT\_TGTCOLORKEY, 27  
 Body functions, 75  
 BOLD  
     oapi::Font, 235  
 BOTTOM  
     oapi::Sketchpad, 337  
 Brush  
     oapi::Brush, 215  
 ButtonLabel  
     MFD, 303  
 ButtonMenu  
     MFD, 303  
 Camera  
     oapiCameraAperture, 97  
     oapiCameraAttach, 97  
     oapiCameraAzimuth, 98  
     oapiCameraGlobalDir, 98  
     oapiCameraGlobalPos, 98  
     oapiCameraInternal, 98  
     oapiCameraMode, 99  
     oapiCameraPolar, 99  
     oapiCameraRotAzimuth, 99  
     oapiCameraRotPolar, 99  
     oapiCameraScaleDist, 100  
     oapiCameraSetAperture, 100  
     oapiCameraSetCockpitDir, 100  
     oapiCameraTarget, 101  
     oapiCameraTargetDist, 101  
     oapiCockpitMode, 101  
 Camera functions, 96  
 CELBODY, 216  
     bEphemeris, 217  
     clbkAtmParam, 219  
     clbkEphemeris, 217  
     clbkFastEphemeris, 218  
     clbkInit, 217  
     Version, 217  
 CELBODY2, 220  
     ~CELBODY2, 222  
 CELBODY2, 222  
     clbkInit, 222  
     FreeAtmosphere, 224  
     FreeAtmosphereModule, 225  
     GetAtmosphere, 223  
     GetChild, 222  
     GetParent, 222  
     LegacyAtmosphereInterface, 223  
     LoadAtmosphereModule, 224  
     SetAtmosphere, 223  
     SidRotPeriod, 223  
 CENTER  
     oapi::Sketchpad, 337  
 cfgprm  
     CFGPRM\_AMBIENTLEVEL, 23  
     CFGPRM\_ATMFOG, 23  
     CFGPRM\_ATMHAZE, 23  
     CFGPRM\_CLOUDS, 23  
     CFGPRM\_CLOUDSHADOWS, 23  
     CFGPRM\_CSPHEREINTENS, 24  
     CFGPRM\_CSPHERETEXTURE, 24  
     CFGPRM\_LOCALLIGHT, 24  
     CFGPRM\_MAXLIGHT, 24  
     CFGPRM\_OBJECTSHADOWS, 24  
     CFGPRM\_OBJECTSPECULAR, 24  
     CFGPRM\_PLANETARIUMFLAG, 24  
     CFGPRM\_STARRENDERPRM, 25  
     CFGPRM\_SURFACELIGHTBRT, 25  
     CFGPRM\_SURFACELIGHTS, 25  
     CFGPRM\_SURFACEMAXLEVEL, 25  
     CFGPRM\_SURFACEREFLECT, 25  
     CFGPRM\_SURFACERIPPLE, 25  
     CFGPRM\_VESSELSHADOWS, 25  
     CFGPRM\_AMBIENTLEVEL  
     cfgprm, 23

CFGPRM\_ATMFOG  
cfgprm, 23  
CFGPRM\_ATMHAZE  
cfgprm, 23  
CFGPRM\_CLOUDS  
cfgprm, 23  
CFGPRM\_CLOUDSHADOWS  
cfgprm, 23  
CFGPRM\_CSHEREINTENS  
cfgprm, 24  
CFGPRM\_CSHERETEXTURE  
cfgprm, 24  
CFGPRM\_LOCALLIGHT  
cfgprm, 24  
CFGPRM\_MAXLIGHT  
cfgprm, 24  
CFGPRM\_OBJECTSHADOWS  
cfgprm, 24  
CFGPRM\_OBJECTSPECULAR  
cfgprm, 24  
CFGPRM\_PLANETARIUMFLAG  
cfgprm, 24  
CFGPRM\_STARRENDERPRM  
cfgprm, 25  
CFGPRM\_SURFACELIGHTBRT  
cfgprm, 25  
CFGPRM\_SURFACELIGHTS  
cfgprm, 25  
CFGPRM\_SURFACEMAXLEVEL  
cfgprm, 25  
CFGPRM\_SURFACEREFLECT  
cfgprm, 25  
CFGPRM\_SURFACERIPPLE  
cfgprm, 25  
CFGPRM\_VESSELSHADOWS  
cfgprm, 25  
clbkADCtrlMode  
VESSEL2, 512  
clbkAnimate  
VESSEL2, 514  
clbkAtmParam  
CELBODY, 219  
clbkBeginBltGroup  
oapi::GraphicsClient, 263  
clbkBlt  
oapi::GraphicsClient, 261, 262  
clbkCloseSession  
oapi::GraphicsClient, 270  
clbkConstants  
ATMOSPHERE, 210  
clbkConsumeBufferedKey  
VESSEL2, 515  
clbkConsumeDirectKey  
VESSEL2, 515  
clbkCopyBitmap  
oapi::GraphicsClient, 265  
clbkCreateAnnotation  
oapi::GraphicsClient, 251  
clbkCreateBrush  
oapi::GraphicsClient, 268  
clbkCreateExhaustStream  
oapi::GraphicsClient, 250  
clbkCreateFont  
oapi::GraphicsClient, 266  
clbkCreateParticleStream  
oapi::GraphicsClient, 249  
clbkCreatePen  
oapi::GraphicsClient, 267  
clbkCreateReentryStream  
oapi::GraphicsClient, 251  
clbkCreateRenderWindow  
oapi::GraphicsClient, 270  
clbkCreateSurface  
oapi::GraphicsClient, 257, 258  
clbkCreateSurfaceEx  
oapi::GraphicsClient, 257  
clbkCreateTexture  
oapi::GraphicsClient, 258  
clbkDeleteVessel  
oapi::Module, 314  
clbkDestroyRenderWindow  
oapi::GraphicsClient, 271  
clbkDisplayFrame  
oapi::GraphicsClient, 272  
clbkDockEvent  
VESSEL2, 514  
clbkDrawHUD  
VESSEL2, 511  
VESSEL3, 527  
clbkEditMeshGroup  
oapi::GraphicsClient, 249  
clbkEndBltGroup  
oapi::GraphicsClient, 264  
clbkEphemeris  
CELBODY, 217  
clbkFastEphemeris  
CELBODY, 218  
clbkFillSurface  
oapi::GraphicsClient, 264  
clbkFocusChanged  
ExternMFD, 233  
oapi::Module, 313  
VESSEL2, 508  
clbkFullscreenMode  
oapi::GraphicsClient, 253  
clbkGeneric  
VESSEL3, 526  
clbkGetDeviceColour

oapi::GraphicsClient, 260  
clbkGetMesh  
    oapi::GraphicsClient, 248  
clbkGetRadiationForce  
    VESSEL3, 529  
clbkGetRenderParam  
    oapi::GraphicsClient, 253  
clbkGetSketchpad  
    oapi::GraphicsClient, 265  
clbkGetSurfaceDC  
    oapi::GraphicsClient, 268  
clbkGetSurfaceSize  
    oapi::GraphicsClient, 260  
clbkGetViewportSize  
    oapi::GraphicsClient, 253  
clbkHUDMode  
    VESSEL2, 513  
clbkIncrSurfaceRef  
    oapi::GraphicsClient, 259  
clbkInit  
    CELBODY, 217  
    CELBODY2, 222  
clbkInitialise  
    oapi::GraphicsClient, 244  
clbkLoadGenericCockpit  
    VESSEL2, 516  
clbkLoadPanel  
    VESSEL2, 516  
clbkLoadPanel2D  
    VESSEL3, 527  
clbkLoadStateEx  
    VESSEL2, 507  
clbkLoadSurface  
    oapi::GraphicsClient, 245  
clbkLoadTexture  
    oapi::GraphicsClient, 244  
clbkLoadVC  
    VESSEL2, 518  
clbkMFDMode  
    VESSEL2, 513  
clbkName  
    ATMOSPHERE, 210  
clbkNavMode  
    VESSEL2, 514  
clbkNewVessel  
    oapi::Module, 314  
clbkOpen  
    LaunchpadItem, 285  
clbkPanelMouseEvent  
    VESSEL2, 517  
    VESSEL3, 525  
clbkPanelRedrawEvent  
    VESSEL2, 517  
    VESSEL3, 526  
clbkParams  
    ATMOSPHERE, 211  
clbkPause  
    oapi::Module, 315  
clbkPlaybackEvent  
    VESSEL2, 510  
clbkPostCreation  
    oapi::GraphicsClient, 270  
    VESSEL2, 508  
clbkPostStep  
    oapi::Module, 312  
    VESSEL2, 509  
clbkPreOpenPopup  
    oapi::GraphicsClient, 249  
clbkPreStep  
    oapi::Module, 312  
    VESSEL2, 509  
clbkRCSMode  
    VESSEL2, 512  
clbkRefreshButtons  
    ExternMFD, 233  
clbkRefreshDisplay  
    ExternMFD, 233  
clbkRefreshVideoData  
    oapi::GraphicsClient, 244  
clbkReleaseBrush  
    oapi::GraphicsClient, 268  
clbkReleaseFont  
    oapi::GraphicsClient, 267  
clbkReleasePen  
    oapi::GraphicsClient, 267  
clbkReleaseSketchpad  
    oapi::GraphicsClient, 266  
clbkReleaseSurface  
    oapi::GraphicsClient, 259  
clbkReleaseSurfaceDC  
    oapi::GraphicsClient, 269  
clbkReleaseTexture  
    oapi::GraphicsClient, 245  
clbkRender2DPanel  
    oapi::GraphicsClient, 256  
clbkRenderHUD  
    VESSEL3, 528  
clbkRenderScene  
    oapi::GraphicsClient, 272  
clbkSaveState  
    VESSEL2, 507  
clbkScaleBlt  
    oapi::GraphicsClient, 262  
clbkSetClassCaps  
    VESSEL2, 506  
clbkSetMeshMaterial  
    oapi::GraphicsClient, 246  
clbkSetMeshProperty

oapi::GraphicsClient, 246  
clbkSetMeshTexture  
    oapi::GraphicsClient, 246  
clbkSetStateEx  
    VESSEL2, 508  
clbkSetSurfaceColourKey  
    oapi::GraphicsClient, 260  
clbkSimulationEnd  
    oapi::Module, 312  
clbkSimulationStart  
    oapi::Module, 312  
clbkStoreMeshPersistent  
    oapi::GraphicsClient, 273  
clbkTimeAccChanged  
    oapi::Module, 314  
clbkTimeJump  
    oapi::Module, 313  
clbkUpdate  
    ExternMFD, 233  
    oapi::GraphicsClient, 271  
clbkUseLaunchpadVideoTab  
    oapi::GraphicsClient, 269  
clbkVCMouseEvent  
    VESSEL2, 519  
clbkVCRedrawEvent  
    VESSEL2, 519  
clbkVesselJump  
    oapi::Module, 314  
clbkVisEvent  
    oapi::GraphicsClient, 248  
clbkVisualCreated  
    VESSEL2, 510  
clbkVisualDestroyed  
    VESSEL2, 511  
clbkWriteConfig  
    LaunchpadItem, 285  
ClearAirfoilDefinitions  
    VESSEL, 409  
ClearAttachments  
    VESSEL, 483  
ClearBeacons  
    VESSEL, 495  
ClearControlSurfaceDefinitions  
    VESSEL, 412  
ClearDockDefinitions  
    VESSEL, 478  
ClearLightEmitters  
    VESSEL, 498  
ClearMeshes  
    VESSEL, 460, 501  
ClearPropellantResources  
    VESSEL, 424  
ClearThrusterDefinitions  
    VESSEL, 431  
ClearVariableDragElements  
    VESSEL, 414  
COLOUR4, 225  
Configuration parameter identifiers, 22  
ConsumeButton  
    MFD, 302  
ConsumeKeyBuffered  
    MFD, 302  
ConsumeKeyImmediate  
    MFD, 302  
Control surface axis orientation, 47  
Coordinate transformations, 92  
CopyMeshFromTemplate  
    VESSEL, 466  
Create  
    VESSEL, 503  
CreateAirfoil  
    VESSEL, 405  
CreateAirfoil2  
    VESSEL, 406  
CreateAirfoil3  
    VESSEL, 407  
CreateAnimation  
    VESSEL, 468  
CreateAttachment  
    VESSEL, 482  
CreateControlSurface  
    VESSEL, 409  
CreateControlSurface2  
    VESSEL, 410  
CreateControlSurface3  
    VESSEL, 411  
CreateDock  
    VESSEL, 477  
CreatePropellantResource  
    VESSEL, 423  
CreateThruster  
    VESSEL, 429  
CreateThrusterGroup  
    VESSEL, 440  
CreateVariableDragElement  
    VESSEL, 413  
crossp  
    vec, 34  
Custom MFD mode definition, 156  
Customisation - custom menu, dialogs, 163  
CustomMFD  
    oapiDisableMFDMode, 157  
    oapiGetMFDModeSpecEx, 157  
    oapiRegisterMFDMode, 158  
    oapiUnregisterMFDMode, 158  
DeactivateNavmode  
    VESSEL, 394

Defines and Enumerations, 28  
DefSetState  
    VESSEL, 387  
DefSetStateEx  
    VESSEL, 387  
DelAirfoil  
    VESSEL, 409  
DelAnimation  
    VESSEL, 469  
DelAnimationComponent  
    VESSEL, 470  
DelAttachment  
    VESSEL, 482  
DelBeacon  
    VESSEL, 495  
DelControlSurface  
    VESSEL, 411  
DelDock  
    VESSEL, 477  
DelExhaust  
    VESSEL, 489  
DelExhaustStream  
    VESSEL, 493  
DelLightEmitter  
    VESSEL, 497  
DelMesh  
    VESSEL, 462  
DelPropellantResource  
    VESSEL, 424  
DelThruster  
    VESSEL, 430  
DelThrusterGroup  
    VESSEL, 441, 499  
Description  
    LaunchpadItem, 284  
Detach  
    oapi::ParticleStream, 324  
DetachChild  
    VESSEL, 486  
Dialog  
    oapiAddTitleButton, 164  
    oapiCloseDialog, 164  
    oapiDefDialogProc, 165  
    oapiFindDialog, 165  
    oapiFindLaunchpadItem, 166  
    oapiGetDialogContext, 166  
    oapiOpenDialog, 166  
    oapiOpenDialogEx, 167  
    oapiOpenHelp, 168  
    oapiOpenLaunchpadHelp, 168  
    oapiRegisterCustomCmd, 168  
    oapiRegisterLaunchpadItem, 169  
    oapiUnregisterCustomCmd, 169  
    oapiUnregisterLaunchpadItem, 169  
DIFFUSE  
    PARTICLESTREAMSPEC, 328  
dist  
    vec, 34  
Dock  
    VESSEL, 480  
DockCount  
    VESSEL, 479  
DockingStatus  
    VESSEL, 480  
dotp  
    vec, 34  
Drawing support functions, 146  
DrawSupport  
    oapiCreateBrush, 147  
    oapiCreateFont, 147  
    oapiCreatePen, 148  
    oapiGetDC, 148  
    oapiGetSketchpad, 149  
    oapiReleaseBrush, 149  
    oapiReleaseDC, 150  
    oapiReleaseFont, 150  
    oapiReleasePen, 150  
    oapiReleaseSketchpad, 150  
EditAirfoil  
    VESSEL, 408  
ELEMENTS, 226  
Ellipse  
    oapi::Sketchpad, 344  
EMISSIVE  
    PARTICLESTREAMSPEC, 328  
EnableIDS  
    VESSEL, 454  
EnableTransponder  
    VESSEL, 453  
ENGINE\_ATTITUDE  
    thrusterparam, 46  
ENGINE\_HOVER  
    thrusterparam, 46  
ENGINE\_MAIN  
    thrusterparam, 45  
ENGINE\_RETRO  
    thrusterparam, 45  
ENGINESTATUS, 227  
ENGINETYPE  
    thrusterparam, 45  
Ephemeris data format bitflags, 22  
EXHAUSTSPEC, 228  
ExitModule  
    general\_clbk, 199  
ExternMFD, 229  
    ~ExternMFD, 231  
    Active, 231

clbkFocusChanged, 233  
clbkRefreshButtons, 233  
clbkRefreshDisplay, 233  
clbkUpdate, 233  
ExternMFD, 231  
GetButtonLabel, 232  
GetDisplaySurface, 232  
GetVessel, 231  
Id, 231  
OpenModeHelp, 233  
ProcessButton, 232  
Resize, 233  
SendKey, 233  
SetMode, 233  
SetVessel, 232

File IO Functions, 170  
FileIO  
    oapiCloseFile, 171  
    oapiOpenFile, 172  
    oapiReadItem\_bool, 172  
    oapiReadItem\_float, 173  
    oapiReadItem\_int, 173  
    oapiReadItem\_string, 173  
    oapiReadItem\_vec, 174  
    oapiReadScenario\_nextline, 174  
    oapiSaveScenario, 175  
    oapiWriteItem\_bool, 175  
    oapiWriteItem\_float, 175  
    oapiWriteItem\_int, 176  
    oapiWriteItem\_string, 176  
    oapiWriteItem\_vec, 176  
    oapiWriteLine, 177  
    oapiWriteLog, 177  
    oapiWriteLogV, 177  
    oapiWriteScenario\_float, 178  
    oapiWriteScenario\_int, 178  
    oapiWriteScenario\_string, 178  
    oapiWriteScenario\_vec, 178

FindRange  
    GraphMFD, 280

flag  
    VESSELSTATUS, 533  
    VESSELSTATUS2, 535

FogParam, 233

Font  
    oapi::Font, 235

FRAME\_GLOBAL  
    refframe, 45

FRAME\_HORIZON  
    refframe, 45

FRAME\_LOCAL  
    refframe, 45

FRAME\_REFLOCAL

refframe, 45  
FreeAtmosphere  
    CELBODY2, 224

FreeAtmosphereModule  
    CELBODY2, 225

Functions for planetary bodies, 102

General module callback functions, 199  
general\_clbk  
    ExitModule, 199  
    InitModule, 200

Generic vessel message identifiers, 56

GetADCtrlMode  
    VESSEL, 393

GetAirfoilParam  
    VESSEL, 407

GetAirspeed  
    VESSEL, 404

GetAirspeedVector  
    VESSEL, 404

GetAltitude  
    VESSEL, 399

GetAngularAcc  
    VESSEL, 390

GetAngularMoment  
    VESSEL, 390

GetAngularVel  
    VESSEL, 389

GetAnimPtr  
    VESSEL, 471

GetAOA  
    VESSEL, 405

GetApDist  
    VESSEL, 399

GetArgPer  
    VESSEL, 398

GetAtmDensity  
    VESSEL, 401

GetAtmosphere  
    CELBODY2, 223

GetAtmPressure  
    VESSEL, 402

GetAtmRef  
    VESSEL, 401

GetAtmTemperature  
    VESSEL, 401

GetAttachmentHandle  
    VESSEL, 485

GetAttachmentId  
    VESSEL, 484

GetAttachmentIndex  
    VESSEL, 485

GetAttachmentParams  
    VESSEL, 483

GetAttachmentStatus  
    VESSEL, 484  
GetAttenuation  
    PointLight, 332  
GetAttitudeLinLevel  
    VESSEL, 450  
GetAttitudeMode  
    VESSEL, 448  
GetAttitudeRotLevel  
    VESSEL, 449  
GetBank  
    VESSEL, 400  
GetBankMomentScale  
    VESSEL, 500  
GetBaseShadowGeometry  
    oapi::GraphicsClient, 256  
GetBaseStructures  
    oapi::GraphicsClient, 256  
GetBaseTileList  
    oapi::GraphicsClient, 255  
GetBeacon  
    VESSEL, 496  
GetButtonLabel  
    ExternMFD, 232  
GetCameraDefaultDirection  
    VESSEL, 457  
GetCameraOffset  
    VESSEL, 456  
GetCelestialMarkers  
    oapi::GraphicsClient, 274  
GetCharSize  
    oapi::Sketchpad, 340  
GetChild  
    CELBODY2, 222  
GetClassName  
    VESSEL, 377  
GetClipRadius  
    VESSEL, 380  
GetCOG\_elev  
    VESSEL, 382  
GetConfigParam  
    oapi::GraphicsClient, 254  
GetControlSurfaceLevel  
    VESSEL, 413  
GetCrossSections  
    VESSEL, 383  
GetCW  
    VESSEL, 414  
GetDamageModel  
    VESSEL, 377  
GetDC  
    oapi::Sketchpad, 347  
GetDefaultColour  
    MFD2, 309  
GetDefaultFont  
    MFD2, 308  
GetDefaultPen  
    MFD2, 308  
GetDefaultPropellantResource  
    VESSEL, 428  
GetDevMesh  
    VESSEL, 465  
GetDirection  
    LightEmitter, 291  
GetDirectionRef  
    LightEmitter, 291  
GetDisplaySurface  
    ExternMFD, 232  
GetDockHandle  
    VESSEL, 479  
GetDockParams  
    VESSEL, 479  
GetDockStatus  
    VESSEL, 479  
GetDrag  
    VESSEL, 420  
GetDragVector  
    VESSEL, 421  
GetDynPressure  
    VESSEL, 402  
GetEditorModule  
    VESSEL, 376  
GetElements  
    VESSEL, 395, 396  
GetEmptyMass  
    VESSEL, 381  
GetEnableFocus  
    VESSEL, 378  
GetEquPos  
    VESSEL, 400  
GetExhaustCount  
    VESSEL, 489  
GetExhaustLevel  
    VESSEL, 490  
GetExhaustSpec  
    VESSEL, 489, 490  
GetFlightModel  
    VESSEL, 377  
GetFlightStatus  
    VESSEL, 387  
GetForceVector  
    VESSEL, 422  
GetFuelMass  
    VESSEL, 429  
GetFuelRate  
    VESSEL, 429  
GetGDIFont  
    oapi::Font, 236

GetGlobalOrientation  
    VESSEL, 391  
GetGlobalPos  
    VESSEL, 388  
GetGlobalVel  
    VESSEL, 388  
GetGravityGradientDamping  
    VESSEL, 385  
GetGravityRef  
    VESSEL, 395  
GetGroundspeed  
    VESSEL, 403  
GetGroundspeedVector  
    VESSEL, 403  
GetGroupThruster  
    VESSEL, 443, 444  
GetGroupThrusterCount  
    VESSEL, 443  
GetHandle  
    VESSEL, 376  
GetHeight  
    MFD2, 307  
GetHorizonAirspeedVector  
    VESSEL, 405  
GetIDS  
    VESSEL, 455  
GetISP  
    VESSEL, 439  
GetLift  
    VESSEL, 420  
GetLiftVector  
    VESSEL, 421  
GetLightEmitter  
    VESSEL, 497  
GetLinearMoment  
    VESSEL, 390  
GetMachNumber  
    VESSEL, 402  
GetManualControlLevel  
    VESSEL, 447  
GetMass  
    VESSEL, 388  
GetMaxFuelMass  
    VESSEL, 428  
GetMesh  
    VESSEL, 464  
GetMeshCount  
    VESSEL, 464  
GetMeshName  
    VESSEL, 465  
GetMeshOffset  
    VESSEL, 464  
GetMeshTemplate  
    VESSEL, 465  
GetMeshVisibilityMode  
    VESSEL, 466  
GetMFDSurface  
    oapi::GraphicsClient, 255  
GetModule  
    oapi::ModuleNV, 316  
GetName  
    VESSEL, 376  
GetNavChannel  
    VESSEL, 453  
GetNavCount  
    VESSEL, 452  
GetNavmodeState  
    VESSEL, 395  
GetNavRecv  
    VESSEL, 501  
GetNavRecvFreq  
    VESSEL, 453  
GetNavSource  
    VESSEL, 456  
GetNosewheelSteering  
    VESSEL, 493  
GetParent  
    CELBODY2, 222  
GetPeDist  
    VESSEL, 398  
GetPenumbra  
    SpotLight, 350  
GetPitch  
    VESSEL, 399  
GetPitchMomentScale  
    VESSEL, 417  
GetPMI  
    VESSEL, 384  
GetPopupList  
    oapi::GraphicsClient, 253  
GetPosition  
    LightEmitter, 289  
GetPositionRef  
    LightEmitter, 290  
GetPropellantCount  
    VESSEL, 424  
GetPropellantEfficiency  
    VESSEL, 426  
GetPropellantFlowrate  
    VESSEL, 427  
GetPropellantHandleByIndex  
    VESSEL, 424  
GetPropellantMass  
    VESSEL, 425  
GetPropellantMaxMass  
    VESSEL, 425  
GetRange  
    PointLight, 331

GetRelativePos  
    VESSEL, 389  
GetRelativeVel  
    VESSEL, 389  
GetRotationMatrix  
    VESSEL, 474  
GetRotDrag  
    VESSEL, 416  
GetShipAirspeedVector  
    VESSEL, 405  
GetSimMJD  
    oapi::ModuleNV, 317  
GetSimStep  
    oapi::ModuleNV, 317  
GetSimTime  
    oapi::ModuleNV, 317  
GetSize  
    VESSEL, 378  
GetSlipAngle  
    VESSEL, 405  
GetSMi  
    VESSEL, 397  
GetStatus  
    VESSEL, 386  
GetStatusEx  
    VESSEL, 386  
GetSuperstructureCG  
    VESSEL, 473  
GetSurface  
    oapi::Sketchpad, 347  
GetSurfaceMarkers  
    oapi::GraphicsClient, 274  
GetSurfaceRef  
    VESSEL, 399  
GetTextWidth  
    oapi::Sketchpad, 341  
GetThrusterCount  
    VESSEL, 431  
GetThrusterDir  
    VESSEL, 433  
GetThrusterGroupHandle  
    VESSEL, 442  
GetThrusterGroupLevel  
    VESSEL, 447  
GetThrusterHandleByIndex  
    VESSEL, 431  
GetThrusterIsp  
    VESSEL, 436  
GetThrusterIsp0  
    VESSEL, 435  
GetThrusterLevel  
    VESSEL, 437  
GetThrusterMax  
    VESSEL, 434, 435  
GetThrusterMax0  
    VESSEL, 433  
GetThrusterMoment  
    VESSEL, 439  
GetThrusterRef  
    VESSEL, 432  
GetThrusterResource  
    VESSEL, 431  
GetThrustVector  
    VESSEL, 421  
GetTorqueVector  
    VESSEL, 422  
GetTotalPropellantFlowrate  
    VESSEL, 427  
GetTotalPropellantMass  
    VESSEL, 426  
GetTouchdownPoints  
    VESSEL, 382  
GetTransponder  
    VESSEL, 455  
GetTrimScale  
    VESSEL, 418  
GetUmbra  
    SpotLight, 350  
 GetUserThrusterGroupCount  
    VESSEL, 444  
 GetUserThrusterGroupHandleByIndex  
    VESSEL, 442  
GetVCHUDSurface  
    oapi::GraphicsClient, 255  
GetVCMFDSurface  
    oapi::GraphicsClient, 255  
GetVessel  
    ExternMFD, 231  
GetVideoData  
    oapi::GraphicsClient, 252  
GetWeightVector  
    VESSEL, 420  
GetWheelbrakeLevel  
    VESSEL, 494  
GetWidth  
    MFD2, 307  
GetWingAspect  
    VESSEL, 415  
GetWingEffectiveness  
    VESSEL, 416  
GetYaw  
    VESSEL, 400  
GetYawMomentScale  
    VESSEL, 418  
Global2Local  
    VESSEL, 476  
GlobalRot  
    VESSEL, 474

GraphicsClient  
    oapi::GraphicsClient, 243

GraphMFD, 276  
    AddGraph, 278  
    AddPlot, 278  
    FindRange, 280  
    GraphMFD, 278  
    Plot, 280  
    SetAutoRange, 279  
    SetAutoTicks, 279  
    SetAxisTitle, 279  
    SetRange, 278

GroundContact  
    VESSEL, 392

GROUPEDITSPEC, 280

Handles, 30

HELPCONTEXT, 281

HorizonInvRot  
    VESSEL, 475

HorizonRot  
    VESSEL, 475

HUD mode identifiers, 51

HUD, MFD and panel functions, 133

HUDPARAM, 282

Id  
    ExternMFD, 231

Identifiers for frames of reference, 45

Identifiers for special render surfaces, 31

Identifiers for visual events, 48

IncEngineLevel  
    VESSEL, 499

IncThrusterGroupLevel  
    VESSEL, 445

IncThrusterGroupLevel\_SingleStep  
    VESSEL, 446

IncThrusterLevel  
    VESSEL, 438

IncThrusterLevel\_SingleStep  
    VESSEL, 439

InitModule  
    general\_clbk, 200

InitNavRadios  
    VESSEL, 452

InsertMesh  
    VESSEL, 461, 462

InvalidateButtons  
    MFD, 300

InvalidateDisplay  
    MFD, 300

IsActive  
    LightEmitter, 288

ITALIC

oapi::Font, 235

Keyboard key identifiers, 186

LaunchpadItem, 283  
    clbkOpen, 285  
    clbkWriteConfig, 285  
    Description, 284  
    Name, 284  
    OpenDialog, 284

LaunchpadVideoTab  
    oapi::GraphicsClient, 273

LaunchpadVideoWndProc  
    oapi::GraphicsClient, 252

LEFT  
    oapi::Sketchpad, 337

LegacyAtmosphereInterface  
    CELBODY2, 223

length  
    vec, 35

Level  
    oapi::ParticleStream, 325

LEVELMAP  
    PARTICLESTREAMSPEC, 328

levelmap  
    PARTICLESTREAMSPEC, 328

LIFT\_HORIZONTAL  
    airfoilliftdir, 47

LIFT\_VERTICAL  
    airfoilliftdir, 47

Light beacon shape parameters, 44

LightEmitter, 286  
    Activate, 288  
    GetDirection, 291  
    GetDirectionRef, 291  
    GetPosition, 289  
    GetPositionRef, 290  
    IsActive, 288  
    LightEmitter, 288  
    SetDirection, 290  
    SetDirectionRef, 291  
    SetPosition, 289  
    SetPositionRef, 289  
    ShiftExplicitPosition, 290

LightEmitterCount  
    VESSEL, 497

Line  
    oapi::Sketchpad, 343

LineTo  
    oapi::Sketchpad, 343

Listclbkflag, 44

LISTENTRY, 292

Listentryflag, 44

LoadAtmosphereModule

CELBODY2, 224  
LoadConstellationLines  
    oapi::GraphicsClient, 273  
LoadMeshClbkFunc  
    Mesh, 124  
LoadStars  
    oapi::GraphicsClient, 273  
Local lighting interface, 43  
Local2Global  
    VESSEL, 475  
Local2Rel  
    VESSEL, 476  
Logical key ids, 192  
LTYPE  
    PARTICLESTREAMSPEC, 328  
ltype  
    PARTICLESTREAMSPEC, 328  
LVL\_FLAT  
    PARTICLESTREAMSPEC, 328  
LVL\_LIN  
    PARTICLESTREAMSPEC, 328  
LVL\_PLIN  
    PARTICLESTREAMSPEC, 328  
LVL\_PSQRT  
    PARTICLESTREAMSPEC, 328  
LVL\_SQRT  
    PARTICLESTREAMSPEC, 328  
Manual control device identifiers, 50  
Manual control mode identifiers, 49  
MATERIAL, 292  
MATRIX3, 293  
Mesh  
    LoadMeshClbkFunc, 124  
    oapiAddMaterial, 124  
    oapiCreateMesh, 124  
    oapiDeleteMaterial, 125  
    oapiDeleteMesh, 125  
    oapiEditMeshGroup, 125  
    oapiGetTextureHandle, 126  
    oapiLoadMesh, 126  
    oapiLoadMeshGlobal, 127  
    oapiLoadTexture, 128  
    oapiMeshGroup, 128  
    oapiMeshGroupCount, 129  
    oapiMeshMaterial, 129  
    oapiMeshMaterialCount, 130  
    oapiMeshTextureCount, 130  
    oapiObjectVisualPtr, 130  
    oapiParticleSetLevelRef, 131  
    oapiReleaseTexture, 131  
    oapiSetMaterial, 131  
    oapiSetMeshProperty, 132  
    oapiSetTexture, 133  
Mesh group editing flags, 41  
MESHGROUP, 294  
MESHGROUP\_TRANSFORM, 295  
MESHGROUPEX, 296  
MeshgroupTransform  
    VESSEL, 467  
MeshModified  
    VESSEL, 467  
MFD, 297  
    ButtonLabel, 303  
    ButtonMenu, 303  
    ConsumeButton, 302  
    ConsumeKeyBuffered, 302  
    ConsumeKeyImmediate, 302  
    InvalidateButtons, 300  
    InvalidateDisplay, 300  
    MFD, 299  
    ReadStatus, 304  
    RecallStatus, 305  
    SelectDefaultFont, 301  
    SelectDefaultPen, 301  
    StoreStatus, 304  
    Title, 300  
    Update, 300  
    WriteStatus, 304  
MFD identifiers, 52  
MFD mode identifiers, 51  
MFD2, 305  
    GetDefaultColour, 309  
    GetDefaultFont, 308  
    GetDefaultPen, 308  
    GetHeight, 307  
    GetWidth, 307  
    MFD2, 306  
    Title, 307  
    Update, 307  
Module  
    oapi::Module, 311  
ModuleNV  
    oapi::ModuleNV, 316  
Mouse event identifiers, 54  
MoveTo  
    oapi::Sketchpad, 343  
mul  
    vec, 35  
Name  
    LaunchpadItem, 284  
NAVDATA, 318  
Navigation mode identifiers, 49  
Navigation radio transmitter functions, 116  
Navigation radio transmitter types, 57  
NavRadio  
    oapiGetNavChannel, 117

oapiGetNavData, 117  
oapiGetNavDescr, 118  
oapiGetNavFreq, 118  
oapiGetNavPos, 118  
oapiGetNavRange, 119  
oapiGetNavSignal, 119  
oapiGetNavType, 119  
oapiNavInRange, 120  
NonsphericalGravityEnabled  
    VESSEL, 392  
NORMAL  
    oapi::Font, 235  
normalise  
    vec, 35  
normangle  
    OrbiterAPI.h, 591  
NTVERTEX, 319  
  
oapi  
    oapiDebugString, 60  
    oapiGetBarycentre, 60  
    oapiGetCmdLine, 60  
    oapiGetInducedDrag, 60  
    oapiGetModuleVersion, 61  
    oapiGetOrbiterInstance, 61  
    oapiGetOrbiterVersion, 62  
    oapiGetPanelScale, 62  
    oapiGetViewportSize, 62  
    oapiGetWaveDrag, 62  
    oapiRegisterExhaustTexture, 63  
    oapiRegisterGraphicsClient, 64  
    oapiRegisterModule, 64  
    oapiRegisterReentryTexture, 64  
oapi::Brush, 215  
    Brush, 215  
oapi::DrawingTool, 226  
oapi::Font, 234  
    BOLD, 235  
    Font, 235  
    GetGDIFont, 236  
    ITALIC, 235  
    NORMAL, 235  
    Style, 235  
    UNDERLINE, 235  
oapi::GraphicsClient, 236  
    ~GraphicsClient, 243  
    clbkBeginBltGroup, 263  
    clbkBlt, 261, 262  
    clbkCloseSession, 270  
    clbkCopyBitmap, 265  
    clbkCreateAnnotation, 251  
    clbkCreateBrush, 268  
    clbkCreateExhaustStream, 250  
    clbkCreateFont, 266  
    clbkCreateParticleStream, 249  
    clbkCreatePen, 267  
    clbkCreateReentryStream, 251  
    clbkCreateRenderWindow, 270  
    clbkCreateSurface, 257, 258  
    clbkCreateSurfaceEx, 257  
    clbkCreateTexture, 258  
    clbkDestroyRenderWindow, 271  
    clbkDisplayFrame, 272  
    clbkEditMeshGroup, 249  
    clbkEndBltGroup, 264  
    clbkFillSurface, 264  
    clbkFullscreenMode, 253  
    clbkGetDeviceColour, 260  
    clbkGetMesh, 248  
    clbkGetRenderParam, 253  
    clbkGetSketchpad, 265  
    clbkGetSurfaceDC, 268  
    clbkGetSurfaceSize, 260  
    clbkGetViewportSize, 253  
    clbkIncrSurfaceRef, 259  
    clbkInitialise, 244  
    clbkLoadSurface, 245  
    clbkLoadTexture, 244  
    clbkPostCreation, 270  
    clbkPreOpenPopup, 249  
    clbkRefreshVideoData, 244  
    clbkReleaseBrush, 268  
    clbkReleaseFont, 267  
    clbkReleasePen, 267  
    clbkReleaseSketchpad, 266  
    clbkReleaseSurface, 259  
    clbkReleaseSurfaceDC, 269  
    clbkReleaseTexture, 245  
    clbkRender2DPanel, 256  
    clbkRenderScene, 272  
    clbkScaleBlt, 262  
    clbkSetMeshMaterial, 246  
    clbkSetMeshProperty, 246  
    clbkSetMeshTexture, 246  
    clbkSetSurfaceColourKey, 260  
    clbkStoreMeshPersistent, 273  
    clbkUpdate, 271  
    clbkUseLaunchpadVideoTab, 269  
    clbkVisEvent, 248  
    GetBaseShadowGeometry, 256  
    GetBaseStructures, 256  
    GetBaseTileList, 255  
    GetCelestialMarkers, 274  
    GetConfigParam, 254  
    GetMFDSurface, 255  
    GetPopupList, 253  
    GetSurfaceMarkers, 274  
    GetVCHUDSurface, 255

GetVCMFDSurface, 255  
GetVideoData, 252  
GraphicsClient, 243  
LaunchpadVideoTab, 273  
LaunchpadVideoWndProc, 252  
LoadConstellationLines, 273  
LoadStars, 273  
RegisterVisObject, 247  
Render2DOverlay, 272  
RenderWndProc, 252  
TexturePath, 254  
UnregisterVisObject, 248  
oapi::GraphicsClient::LABELLIST, 275  
oapi::GraphicsClient::VIDEODATA, 275  
oapi::IVECTOR2, 282  
oapi::Module, 310  
    clbkDeleteVessel, 314  
    clbkFocusChanged, 313  
    clbkNewVessel, 314  
    clbkPause, 315  
    clbkPostStep, 312  
    clbkPreStep, 312  
    clbkSimulationEnd, 312  
    clbkSimulationStart, 312  
    clbkTimeAccChanged, 314  
    clbkTimeJump, 313  
    clbkVesselJump, 314  
    Module, 311  
    RENDER\_FULLSCREEN, 311  
    RENDER\_NONE, 311  
    RENDER\_WINDOW, 311  
    RenderMode, 311  
oapi::ModuleNV, 315  
    GetModule, 316  
    GetSimMJD, 317  
    GetSimStep, 317  
    GetSimTime, 317  
    ModuleNV, 316  
    Version, 316  
oapi::ParticleStream, 321  
    Attach, 323, 324  
    Detach, 324  
    Level, 325  
    ParticleStream, 323  
    SetFixedDir, 324  
    SetFixedPos, 324  
    SetLevelPtr, 325  
    SetVariableDir, 325  
    SetVariablePos, 325  
oapi::Pen, 328  
    Pen, 329  
oapi::ScreenAnnotation, 332  
    ScreenAnnotation, 334  
    SetColour, 334  
                SetPosition, 334  
                SetSize, 334  
                SetText, 334  
oapi::Sketchpad, 335  
    BASELINE, 337  
    BK\_OPAQUE, 338  
    BK\_TRANSPARENT, 338  
    BkgMode, 337  
    BOTTOM, 337  
    CENTER, 337  
    Ellipse, 344  
    GetCharSize, 340  
    GetDC, 347  
    GetSurface, 347  
    GetTextWidth, 341  
    LEFT, 337  
    Line, 343  
    LineTo, 343  
    MoveTo, 343  
    Pixel, 342  
    Polygon, 345  
    Polyline, 345  
    PolyPolygon, 346  
    PolyPolyline, 346  
    Rectangle, 344  
    RIGHT, 337  
    SetBackgroundColor, 339  
    SetBackgroundMode, 340  
    SetBrush, 339  
    SetFont, 338  
    SetOrigin, 341  
    SetPen, 338  
    SetTextAlign, 339  
    SetTextColor, 339  
    Sketchpad, 338  
    TAlign\_horizontal, 337  
    TAlign\_vertical, 337  
    Text, 341  
    TextBox, 342  
    TOP, 337  
oapi\_body  
    oapiGetGlobalPos, 75  
    oapiGetGlobalVel, 75  
    oapiGetMass, 76  
    oapiGetRelativePos, 76  
    oapiGetRelativeVel, 76  
    oapiGetSize, 77  
oapi\_time  
    oapiGetFrameRate, 112  
    oapiGetPause, 112  
    oapiGetSimMJD, 113  
    oapiGetSimStep, 113  
    oapiGetSimTime, 113  
    oapiGetSysMJD, 113

oapiGetSysStep, 114  
oapiGetSysTime, 114  
oapiGetTimeAcceleration, 114  
oapiSetPause, 115  
oapiSetSimMJD, 115  
oapiSetTimeAcceleration, 116  
oapiTime2MJD, 116  
oapi\_transformation  
    oapiEquToGlobal, 93  
    oapiEquToLocal, 93  
    oapiGetRotationMatrix, 93  
    oapiGlobalToEqu, 94  
    oapiGlobalToLocal, 94  
    oapiLocalToEqu, 95  
    oapiLocalToGlobal, 95  
    oapiOrthodome, 95  
oapi\_vessel  
    oapiGetAirspeed, 79  
    oapiGetAirspeedVector, 80  
    oapiGetAltitude, 80  
    oapiGetAtm, 81  
    oapiGetAttitudeMode, 81  
    oapiGetBank, 81  
    oapiGetDockHandle, 82  
    oapiGetDockStatus, 82  
    oapiGetEmptyMass, 82  
    oapiGetEngineStatus, 83  
    oapiGetEquPos, 83  
    oapiGetFocusAltitude, 84  
    oapiGetFocusAttitudeMode, 84  
    oapiGetFocusBank, 84  
    oapiGetFocusEngineStatus, 84  
    oapiGetFocusEquPos, 84  
    oapiGetFocusGlobalPos, 85  
    oapiGetFocusGlobalVel, 85  
    oapiGetFocusHeading, 85  
    oapiGetFocusPitch, 86  
    oapiGetFocusRelativePos, 86  
    oapiGetFocusRelativeVel, 86  
    oapiGetFuelMass, 87  
    oapiGetGroundspeed, 87  
    oapiGetGroundspeedVector, 87  
    oapiGetHeading, 88  
    oapiGetMaxFuelMass, 88  
    oapiGetPitch, 89  
    oapiGetPropellantHandle, 89  
    oapiGetPropellantMass, 89  
    oapiGetPropellantMaxMass, 90  
    oapiSetAttitudeMode, 90  
    oapiSetEmptyMass, 90  
    oapiSetEngineLevel, 91  
    oapiSetFocusAttitudeMode, 91  
    oapiToggleAttitudeMode, 91  
    oapiToggleFocusAttitudeMode, 92  
oapiAddMaterial  
    Mesh, 124  
oapiAddTitleButton  
    Dialog, 164  
oapiAnnotationSetColour  
    Annotations, 181  
oapiAnnotationSetPos  
    Annotations, 181  
oapiAnnotationSetSize  
    Annotations, 182  
oapiAnnotationSetText  
    Annotations, 182  
oapiAsyncScriptCmd  
    Script, 121  
oapiBeginBltGroup  
    Surface, 152  
oapiBlt  
    Surface, 152, 153  
oapiBltPanelAreaBackground  
    Panel, 135  
oapiCameraAperture  
    Camera, 97  
oapiCameraAttach  
    Camera, 97  
oapiCameraAzimuth  
    Camera, 98  
oapiCameraGlobalDir  
    Camera, 98  
oapiCameraGlobalPos  
    Camera, 98  
oapiCameraInternal  
    Camera, 98  
oapiCameraMode  
    Camera, 99  
oapiCameraPolar  
    Camera, 99  
oapiCameraRotAzimuth  
    Camera, 99  
oapiCameraRotPolar  
    Camera, 99  
oapiCameraScaleDist  
    Camera, 100  
oapiCameraSetAperture  
    Camera, 100  
oapiCameraSetCockpitDir  
    Camera, 100  
oapiCameraTarget  
    Camera, 101  
oapiCameraTargetDist  
    Camera, 101  
oapiClearSurfaceColourKey  
    Surface, 153  
oapiCloseDialog  
    Dialog, 164

oapiCloseFile  
    FileIO, 171  
oapiCockpitMode  
    Camera, 101  
oapiColourFill  
    Surface, 154  
oapiCreateAnnotation  
    Annotations, 182  
oapiCreateBrush  
    DrawSupport, 147  
oapiCreateFont  
    DrawSupport, 147  
oapiCreateInterpreter  
    Script, 121  
oapiCreateMesh  
    Mesh, 124  
oapiCreatePen  
    DrawSupport, 148  
oapiCreateSurface  
    Surface, 154, 155  
oapiCreateTextureSurface  
    Surface, 155  
oapiCreateVessel  
    VesselCreation, 73  
oapiCreateVesselEx  
    VesselCreation, 73  
oapiDebugString  
    oapi, 60  
oapiDecHUDIntensity  
    Panel, 136  
oapiDefDialogProc  
    Dialog, 165  
oapiDelAnnotation  
    Annotations, 182  
oapiDeleteMaterial  
    Mesh, 125  
oapiDeleteMesh  
    Mesh, 125  
oapiDeleteVessel  
    VesselCreation, 74  
oapiDelInterpreter  
    Script, 121  
oapiDestroySurface  
    Surface, 155  
oapiDisableMFDMode  
    CustomMFD, 157  
oapiEditMeshGroup  
    Mesh, 125  
oapiEndBltGroup  
    Surface, 156  
oapiEquToGlobal  
    oapi\_transformation, 93  
oapiEquToLocal  
    oapi\_transformation, 93  
oapiExecScriptCmd  
    Script, 121  
oapiFindDialog  
    Dialog, 165  
oapiFindLaunchpadItem  
    Dialog, 166  
oapiGetAirspeed  
    oapi\_vessel, 79  
oapiGetAirspeedVector  
    oapi\_vessel, 80  
    Obsolete, 184  
oapiGetAltitude  
    oapi\_vessel, 80  
oapiGetAtm  
    oapi\_vessel, 81  
oapiGetAtmPressureDensity  
    Obsolete, 184  
oapiGetAttitudeMode  
    oapi\_vessel, 81  
oapiGetBank  
    oapi\_vessel, 81  
oapiGetBarycentre  
    oapi, 60  
oapiGetBaseByIndex  
    ObjectAccess, 66  
oapiGetBaseByName  
    ObjectAccess, 66  
oapiGetBaseCount  
    ObjectAccess, 66  
oapiGetBaseEquPos  
    BaseInterface, 109  
oapiGetBasePadCount  
    BaseInterface, 110  
oapiGetBasePadEquPos  
    BaseInterface, 110  
oapiGetBasePadNav  
    BaseInterface, 110  
oapiGetBasePadStatus  
    BaseInterface, 111  
oapiGetBasePlanet  
    BaseInterface, 111  
oapiGetCelbodyInterface  
    ObjectAccess, 67  
oapiGetCmdLine  
    oapi, 60  
oapiGetColour  
    Utility, 179  
oapiGetDC  
    DrawSupport, 148  
oapiGetDialogContext  
    Dialog, 166  
oapiGetDockHandle  
    oapi\_vessel, 82  
oapiGetDockStatus

oapi\_vessel, 82  
oapiGetEmptyMass  
    oapi\_vessel, 82  
oapiGetEngineStatus  
    oapi\_vessel, 83  
oapiGetEquPos  
    oapi\_vessel, 83  
oapiGetFocusAirspeed  
    Obsolete, 184  
oapiGetFocusAirspeedVector  
    Obsolete, 184  
oapiGetFocusAltitude  
    oapi\_vessel, 84  
oapiGetFocusAtmPressureDensity  
    Obsolete, 184  
oapiGetFocusAttitudeMode  
    oapi\_vessel, 84  
oapiGetFocusBank  
    oapi\_vessel, 84  
oapiGetFocusEngineStatus  
    oapi\_vessel, 84  
oapiGetFocusEquPos  
    oapi\_vessel, 84  
oapiGetFocusGlobalPos  
    oapi\_vessel, 85  
oapiGetFocusGlobalVel  
    oapi\_vessel, 85  
oapiGetFocusHeading  
    oapi\_vessel, 85  
oapiGetFocusInterface  
    ObjectAccess, 67  
oapiGetFocusObject  
    ObjectAccess, 67  
oapiGetFocusPitch  
    oapi\_vessel, 86  
oapiGetFocusRelativePos  
    oapi\_vessel, 86  
oapiGetFocusRelativeVel  
    oapi\_vessel, 86  
oapiGetFocusShipAirspeedVector  
    Obsolete, 185  
oapiGetFrameRate  
    oapi\_time, 112  
oapiGetFuelMass  
    oapi\_vessel, 87  
oapiGetGbodyByIndex  
    ObjectAccess, 67  
oapiGetGbodyByName  
    ObjectAccess, 68  
oapiGetGbodyCount  
    ObjectAccess, 68  
oapiGetGlobalPos  
    oapi\_body, 75  
oapiGetGlobalVel  
    oapi\_body, 75  
oapiGetGroundspeed  
    oapi\_vessel, 87  
oapiGetGroundspeedVector  
    oapi\_vessel, 87  
oapiGetGroundVector  
    Planet, 103  
oapiGetHeading  
    oapi\_vessel, 88  
oapiGetHUDMode  
    Panel, 136  
oapiGetInducedDrag  
    oapi, 60  
oapiGetMass  
    oapi\_body, 76  
oapiGetMaxFuelMass  
    oapi\_vessel, 88  
oapiGetMFDMode  
    Panel, 136  
oapiGetMFDModeSpec  
    Obsolete, 185  
oapiGetMFDModeSpecEx  
    CustomMFD, 157  
oapiGetModuleVersion  
    oapi, 61  
oapiGetNavChannel  
    NavRadio, 117  
oapiGetNavData  
    NavRadio, 117  
oapiGetNavDescr  
    NavRadio, 118  
oapiGetNavFreq  
    NavRadio, 118  
oapiGetNavPos  
    NavRadio, 118  
oapiGetNavRange  
    NavRadio, 119  
oapiGetNavSignal  
    NavRadio, 119  
oapiGetNavType  
    NavRadio, 119  
oapiGetObjectByIndex  
    ObjectAccess, 68  
oapiGetObjectByName  
    ObjectAccess, 69  
oapiGetObjectCount  
    ObjectAccess, 69  
oapiGetObjectName  
    ObjectAccess, 69  
oapiGetObjectParam  
    ObjectAccess, 70  
oapiGetObjectType  
    ObjectAccess, 70  
oapiGetOrbiterInstance

oapi, 61  
oapiGetOrbiterVersion  
    oapi, 62  
oapiGetPanelScale  
    oapi, 62  
oapiGetPause  
    oapi\_time, 112  
oapiGetPitch  
    oapi\_vessel, 89  
oapiGetPlanetAtmConstants  
    Planet, 103  
oapiGetPlanetAtmParams  
    Planet, 104  
oapiGetPlanetCurrentRotation  
    Planet, 105  
oapiGetPlanetJCoeff  
    Planet, 105  
oapiGetPlanetJCoeffCount  
    Planet, 106  
oapiGetPlanetObliquity  
    Planet, 106  
oapiGetPlanetObliquityMatrix  
    Planet, 107  
oapiGetPlanetPeriod  
    Planet, 107  
oapiGetPlanetTheta  
    Planet, 107  
oapiGetPropellantHandle  
    oapi\_vessel, 89  
oapiGetPropellantMass  
    oapi\_vessel, 89  
oapiGetPropellantMaxMass  
    oapi\_vessel, 90  
oapiGetRelativePos  
    oapi\_body, 76  
oapiGetRelativeVel  
    oapi\_body, 76  
oapiGetRotationMatrix  
    oapi\_transformation, 93  
oapiGetShipAirspeedVector  
    Obsolete, 185  
oapiGetSimMJD  
    oapi\_time, 113  
oapiGetSimStep  
    oapi\_time, 113  
oapiGetSimTime  
    oapi\_time, 113  
oapiGetSize  
    oapi\_body, 77  
oapiGetSketchpad  
    DrawSupport, 149  
oapiGetStationByIndex  
    Obsolete, 185  
oapiGetStationByName  
    oapi, 61  
oapiGetStationCount  
    Obsolete, 186  
oapiGetSysMJD  
    oapi\_time, 113  
oapiGetSysStep  
    oapi\_time, 114  
oapiGetSysTime  
    oapi\_time, 114  
oapiGetTextureHandle  
    Mesh, 126  
oapiGetTimeAcceleration  
    oapi\_time, 114  
oapiGetVesselByIndex  
    ObjectAccess, 71  
oapiGetVesselByName  
    ObjectAccess, 71  
oapiGetVesselCount  
    ObjectAccess, 71  
oapiGetVesselInterface  
    ObjectAccess, 72  
oapiGetViewportSize  
    oapi, 62  
oapiGetWaveDrag  
    oapi, 62  
oapiGetWindVector  
    Planet, 108  
oapiGlobalToEqu  
    oapi\_transformation, 94  
oapiGlobalToLocal  
    oapi\_transformation, 94  
oapiIncHUDIntensity  
    Panel, 137  
oapiIsVessel  
    ObjectAccess, 72  
oapiLoadMesh  
    Mesh, 126  
oapiLoadMeshGlobal  
    Mesh, 127  
oapiLoadTexture  
    Mesh, 128  
oapiLocalToEqu  
    oapi\_transformation, 95  
oapiLocalToGlobal  
    oapi\_transformation, 95  
oapiMeshGroup  
    Mesh, 128  
oapiMeshGroupCount  
    Mesh, 129  
oapiMeshMaterial  
    Mesh, 129  
oapiMeshMaterialCount  
    Mesh, 130  
oapiMeshTextureCount

Mesh, 130  
oapiMFDButtonLabel  
    Panel, 137  
oapiNavInRange  
    NavRadio, 120  
oapiObjectVisualPtr  
    Mesh, 130  
oapiOpenDialog  
    Dialog, 166  
oapiOpenDialogEx  
    Dialog, 167  
oapiOpenFile  
    FileIO, 172  
oapiOpenHelp  
    Dialog, 168  
oapiOpenInputBox  
    UserInput, 180  
oapiOpenLaunchpadHelp  
    Dialog, 168  
oapiOpenMFD  
    Panel, 137  
oapiOrthodome  
    oapi\_transformation, 95  
oapiParticleSetLevelRef  
    Mesh, 131  
oapiPlanetHasAtmosphere  
    Planet, 108  
oapiProcessMFDButton  
    Panel, 138  
oapiRand  
    Utility, 179  
oapiReadItem\_bool  
    FileIO, 172  
oapiReadItem\_float  
    FileIO, 173  
oapiReadItem\_int  
    FileIO, 173  
oapiReadItem\_string  
    FileIO, 173  
oapiReadItem\_vec  
    FileIO, 174  
oapiReadScenario\_nextline  
    FileIO, 174  
oapiRefreshMFDButtons  
    Panel, 138  
oapiRegisterCustomCmd  
    Dialog, 168  
oapiRegisterExhaustTexture  
    oapi, 63  
oapiRegisterGraphicsClient  
    oapi, 64  
oapiRegisterLaunchpadItem  
    Dialog, 169  
oapiRegisterMFD  
    Panel, 139  
oapiRegisterMFDMode  
    CustomMFD, 158  
    Obsolete, 186  
oapiRegisterModule  
    oapi, 64  
oapiRegisterPanelArea  
    Panel, 140  
oapiRegisterPanelBackground  
    Panel, 141  
oapiRegisterReentryTexture  
    oapi, 64  
oapiReleaseBrush  
    DrawSupport, 149  
oapiReleaseDC  
    DrawSupport, 150  
oapiReleaseFont  
    DrawSupport, 150  
oapiReleasePen  
    DrawSupport, 150  
oapiReleaseSketchpad  
    DrawSupport, 150  
oapiReleaseTexture  
    Mesh, 131  
oapiRenderHUD  
    Panel, 141  
oapiSaveScenario  
    FileIO, 175  
oapiSendMFDKey  
    Panel, 142  
oapiSetAttitudeMode  
    oapi\_vessel, 90  
oapiSetDefNavDisplay  
    Panel, 142  
oapiSetDefRCSDisplay  
    Panel, 142  
oapiSetEmptyMass  
    oapi\_vessel, 90  
oapiSetEngineLevel  
    oapi\_vessel, 91  
oapiSetFocusAttitudeMode  
    oapi\_vessel, 91  
oapiSetFocusObject  
    ObjectAccess, 72  
oapiSetHUDMode  
    Panel, 143  
oapiSetMaterial  
    Mesh, 131  
oapiSetMeshProperty  
    Mesh, 132  
oapiSetPanel  
    Panel, 144  
oapiSetPanelNeighbours  
    Panel, 144

oapiSetPause  
    oapi\_time, 115  
oapiSetSimMJD  
    oapi\_time, 115  
oapiSetSurfaceColourKey  
    Surface, 156  
oapiSetTexture  
    Mesh, 133  
oapiSetTimeAcceleration  
    oapi\_time, 116  
oapiSwitchPanel  
    Panel, 144  
oapiTime2MJD  
    oapi\_time, 116  
oapiToggleAttitudeMode  
    oapi\_vessel, 91  
oapiToggleFocusAttitudeMode  
    oapi\_vessel, 92  
oapiToggleHudColour  
    Panel, 145  
oapiToggleMFD\_on  
    Panel, 145  
oapiTriggerPanelRedrawArea  
    Panel, 145  
oapiTriggerRedrawArea  
    Panel, 145  
oapiUnregisterCustomCmd  
    Dialog, 169  
oapiUnregisterLaunchpadItem  
    Dialog, 169  
oapiUnregisterMFDMode  
    CustomMFD, 158  
oapiVCRegisterArea  
    VirtualCockpit, 159  
oapiVCRegisterHUD  
    VirtualCockpit, 160  
oapiVCRegisterMFD  
    VirtualCockpit, 160  
oapiVCSetAreaClickmode\_Quadrilateral  
    VirtualCockpit, 161  
oapiVCSetAreaClickmode\_Spherical  
    VirtualCockpit, 161  
oapiVCSetNeighbours  
    VirtualCockpit, 162  
oapiVCTriggerRedrawArea  
    VirtualCockpit, 162  
oapiWriteItem\_bool  
    FileIO, 175  
oapiWriteItem\_float  
    FileIO, 175  
oapiWriteItem\_int  
    FileIO, 176  
oapiWriteItem\_string  
    FileIO, 176

oapiWriteItem\_vec  
    FileIO, 176  
oapiWriteLine  
    FileIO, 177  
oapiWriteLog  
    FileIO, 177  
oapiWriteLogV  
    FileIO, 177  
oapiWriteScenario\_float  
    FileIO, 178  
oapiWriteScenario\_int  
    FileIO, 178  
oapiWriteScenario\_string  
    FileIO, 178  
oapiWriteScenario\_vec  
    FileIO, 178

Object access functions, 65  
Object parameter flags, 57  
ObjectAccess  
    oapiGetBaseByIndex, 66  
    oapiGetBaseByName, 66  
    oapiGetBaseCount, 66  
    oapiGetCelbodyInterface, 67  
    oapiGetFocusInterface, 67  
    oapiGetFocusObject, 67  
    oapiGetGbodyByIndex, 67  
    oapiGetGbodyByName, 68  
    oapiGetGbodyCount, 68  
    oapiGetObjectByIndex, 68  
    oapiGetObjectByName, 69  
    oapiGetObjectCount, 69  
    oapiGetObjectName, 69  
    oapiGetObjectParam, 70  
    oapiGetObjectType, 70  
    oapiGetVesselByIndex, 71  
    oapiGetVesselByName, 71  
    oapiGetVesselCount, 71  
    oapiGetVesselInterface, 72  
    oapiIsVessel, 72  
    oapiSetFocusObject, 72

Obsolete  
    oapiGetAirspeedVector, 184  
    oapiGetAtmPressureDensity, 184  
    oapiGetFocusAirspeed, 184  
    oapiGetFocusAirspeedVector, 184  
    oapiGetFocusAtmPressureDensity, 184  
    oapiGetFocusShipAirspeedVector, 185  
    oapiGetMFDModeSpec, 185  
    oapiGetShipAirspeedVector, 185  
    oapiGetStationByIndex, 185  
    oapiGetStationByName, 186  
    oapiGetStationCount, 186  
    oapiRegisterMFDMode, 186

Obsolete functions, 183

Onscreen annotations, 180  
opcCloseRenderViewport  
    plugin\_clbk, 202  
opcDeleteVessel  
    plugin\_clbk, 202  
opcFocusChanged  
    plugin\_clbk, 203  
opcOpenRenderViewport  
    plugin\_clbk, 203  
opcPause  
    plugin\_clbk, 204  
opcPostStep  
    plugin\_clbk, 204  
opcPreStep  
    plugin\_clbk, 205  
opcTimeAccChanged  
    plugin\_clbk, 205  
OpenDialog  
    LaunchpadItem, 284  
OpenModeHelp  
    ExternMFD, 233  
operator\*  
    vec, 36  
operator\*=  
    vec, 36  
operator+  
    vec, 37  
operator+=  
    vec, 37  
operator-  
    vec, 37, 38  
operator-=  
    vec, 38  
operator/  
    vec, 38, 39  
operator/=  
    vec, 39  
Orbiter API interface methods, 58  
OrbiterAPI.h  
    normangle, 591  
    posangle, 592  
Orbitersdk/include/CelBodyAPI.h, 538  
Orbitersdk/include/DrawAPI.h, 539  
Orbitersdk/include/MFDAPI.h, 540  
Orbitersdk/include/OrbiterAPI.h, 541  
Orbitersdk/include/VesselAPI.h, 592  
ORBITPARAM, 320  
OrbitStabilised  
    VESSEL, 392  
outerp  
    vec, 39  
ovcExit  
    vessel\_clbk, 201  
ovcInit  
    vessel\_clbk, 201  
Panel  
    oapiBltPanelAreaBackground, 135  
    oapiDecHUDIntensity, 136  
    oapiGetHUDMode, 136  
    oapiGetMFDMode, 136  
    oapiIncHUDIntensity, 137  
    oapiMFDButtonLabel, 137  
    oapiOpenMFD, 137  
    oapiProcessMFDButton, 138  
    oapiRefreshMFDButtons, 138  
    oapiRegisterMFD, 139  
    oapiRegisterPanelArea, 140  
    oapiRegisterPanelBackground, 141  
    oapiRenderHUD, 141  
    oapiSendMFDFKey, 142  
    oapiSetDefNavDisplay, 142  
    oapiSetDefRCSDisplay, 142  
    oapiSetHUDMode, 143  
    oapiSetPanel, 144  
    oapiSetPanelNeighbours, 144  
    oapiSwitchPanel, 144  
    oapiToggleHudColour, 145  
    oapiToggleMFD\_on, 145  
    oapiTriggerPanelRedrawArea, 145  
    oapiTriggerRedrawArea, 145  
Panel area texture mapping identifiers, 55  
Panel neighbour identifiers, 53  
Panel redraw event identifiers, 54  
ParseScenarioLine  
    VESSEL, 503  
ParseScenarioLineEx  
    VESSEL, 498  
ParticleStream  
    oapi::ParticleStream, 323  
PARTICLESTREAMSPEC, 326  
    DIFFUSE, 328  
    EMISSIVE, 328  
    LEVELMAP, 328  
    levelmap, 328  
    LTYPE, 328  
    ltype, 328  
    LVL\_FLAT, 328  
    LVL\_LIN, 328  
    LVL\_PLIN, 328  
    LVL\_PSQRT, 328  
    LVL\_SQRT, 328  
Pen  
    oapi::Pen, 329  
Pixel  
    oapi::Sketchpad, 342  
Planet  
    oapiGetGroundVector, 103

oapiGetPlanetAtmConstants, 103  
oapiGetPlanetAtmParams, 104  
oapiGetPlanetCurrentRotation, 105  
oapiGetPlanetJCoeff, 105  
oapiGetPlanetJCoeffCount, 106  
oapiGetPlanetObliquity, 106  
oapiGetPlanetObliquityMatrix, 107  
oapiGetPlanetPeriod, 107  
oapiGetPlanetTheta, 107  
oapiGetWindVector, 108  
oapiPlanetHasAtmosphere, 108  
**Playback**  
    VESSEL, 472  
**Plot**  
    GraphMFD, 280  
**Plugin module callback functions**, 201  
**plugin\_clbk**  
    opcCloseRenderViewport, 202  
    opcDeleteVessel, 202  
    opcFocusChanged, 203  
    opcOpenRenderViewport, 203  
    opcPause, 204  
    opcPostStep, 204  
    opcPreStep, 205  
    opcTimeAccChanged, 205  
**PointLight**, 329  
    GetAttenuation, 332  
    GetRange, 331  
    PointLight, 331  
    SetAttenuation, 332  
    SetRange, 331  
**Polygon**  
    oapi::Sketchpad, 345  
**Polyline**  
    oapi::Sketchpad, 345  
**PolyPolygon**  
    oapi::Sketchpad, 346  
**PolyPolyline**  
    oapi::Sketchpad, 346  
**posangle**  
    OrbiterAPI.h, 592  
**PRM\_ALT**  
    ATMOSPHERE, 210  
**PRM\_AP**  
    ATMOSPHERE, 210  
**PRM\_F**  
    ATMOSPHERE, 210  
**PRM\_FBR**  
    ATMOSPHERE, 210  
**PRM\_LAT**  
    ATMOSPHERE, 210  
**PRM\_LNG**  
    ATMOSPHERE, 210  
**PRM\_IN\_FLAG**

    ATMOSPHERE, 210  
    ExternMFD, 233  
    ProcessButton  
        ExternMFD, 232  
    RCS mode identifiers, 50  
    ReadStatus  
        MFD, 304  
    RecallStatus  
        MFD, 305  
    RecordEvent  
        VESSEL, 472  
    Recording  
        VESSEL, 471  
    Rectangle  
        oapi::Sketchpad, 344  
    REFFRAME  
        refframe, 45  
    refframe  
        FRAME\_GLOBAL, 45  
        FRAME\_HORIZON, 45  
        FRAME\_LOCAL, 45  
        FRAME\_REFLOCAL, 45  
        REFFRAME, 45  
    RegisterAnimation  
        VESSEL, 468  
    RegisterPanelArea  
        VESSEL3, 524  
        VESSEL4, 530  
    RegisterPanelMFDGeometry  
        VESSEL3, 523  
    RegisterVisObject  
        oapi::GraphicsClient, 247  
    Render parameter identifiers, 26  
    Render2DOverlay  
        oapi::GraphicsClient, 272  
    RENDER\_FULLSCREEN  
        oapi::Module, 311  
    RENDER\_NONE  
        oapi::Module, 311  
    RENDER\_WINDOW  
        oapi::Module, 311  
    RenderMode  
        oapi::Module, 311  
    renderprm  
        RP\_ISTLDEVICE, 26  
    RENDERTGT\_MAINWINDOW  
        surfid, 31  
    RenderWndProc  
        oapi::GraphicsClient, 252  
    Resize  
        ExternMFD, 233  
    RIGHT  
        oapi::Sketchpad, 337  
    RP\_ISTLDEVICE

renderprm, 26  
SaveDefaultState  
    VESSEL, 502  
ScreenAnnotation  
    oapi::ScreenAnnotation, 334  
Script  
    oapiAsyncScriptCmd, 121  
    oapiCreateInterpreter, 121  
    oapiDelInterpreter, 121  
    oapiExecScriptCmd, 121  
Script interpreter functions, 120  
SelectDefaultFont  
    MFD, 301  
SelectDefaultPen  
    MFD, 301  
SendBufferedKey  
    VESSEL, 451  
SendKey  
    ExternMFD, 233  
SetADCtrlMode  
    VESSEL, 393  
SetAlbedoRGB  
    VESSEL, 380  
SetAngularVel  
    VESSEL, 391  
SetAnimation  
    VESSEL, 471  
SetAperture  
    SpotLight, 350  
SetAtmosphere  
    CELBODY2, 223  
SetAttachmentParams  
    VESSEL, 483  
SetAttenuation  
    PointLight, 332  
SetAttitudeLinLevel  
    VESSEL, 451  
SetAttitudeMode  
    VESSEL, 448  
SetAttitudeRotLevel  
    VESSEL, 450  
SetAutoRange  
    GraphMFD, 279  
SetAutoTicks  
    GraphMFD, 279  
SetAxisTitle  
    GraphMFD, 279  
SetBackgroundColor  
    oapi::Sketchpad, 339  
SetBackgroundMode  
    oapi::Sketchpad, 340  
SetBankMomentScale  
    VESSEL, 500  
SetBrush  
    oapi::Sketchpad, 339  
SetCameraCatchAngle  
    VESSEL, 458  
SetCameraDefaultDirection  
    VESSEL, 457  
SetCameraMovement  
    VESSEL, 459  
SetCameraOffset  
    VESSEL, 456  
SetCameraRotationRange  
    VESSEL, 458  
SetCameraShiftRange  
    VESSEL, 459  
SetClipRadius  
    VESSEL, 381  
SetCOG\_elev  
    VESSEL, 501  
SetColour  
    oapi::ScreenAnnotation, 334  
SetControlSurfaceLevel  
    VESSEL, 412  
SetCrossSections  
    VESSEL, 384  
SetCW  
    VESSEL, 414  
SetDefaultPropellantResource  
    VESSEL, 428  
SetDirection  
    LightEmitter, 290  
SetDirectionRef  
    LightEmitter, 291  
SetDockMode  
    VESSEL, 481  
SetDockParams  
    VESSEL, 478  
SetElements  
    VESSEL, 397  
SetEmptyMass  
    VESSEL, 381  
SetEnableFocus  
    VESSEL, 378  
SetEngineLevel  
    VESSEL, 498  
SetExhaustScales  
    VESSEL, 499  
SetFixedDir  
    oapi::ParticleStream, 324  
SetFixedPos  
    oapi::ParticleStream, 324  
SetFont  
    oapi::Sketchpad, 338  
SetFuelMass  
    VESSEL, 429

SetGlobalOrientation  
    VESSEL, 391  
SetGravityGradientDamping  
    VESSEL, 385  
SetIDSChannel  
    VESSEL, 454  
SetISP  
    VESSEL, 440  
SetLevelPtr  
    oapi::ParticleStream, 325  
SetLiftCoeffFunc  
    VESSEL, 419  
SetMaxFuelMass  
    VESSEL, 428  
SetMaxWheelbrakeForce  
    VESSEL, 494  
SetMeshVisibilityMode  
    VESSEL, 466  
SetMeshVisibleInternal  
    VESSEL, 502  
SetMode  
    ExternMFD, 233  
SetNavChannel  
    VESSEL, 452  
SetNavRecv  
    VESSEL, 501  
SetNosewheelSteering  
    VESSEL, 493  
SetOrigin  
    oapi::Sketchpad, 341  
SetPanelBackground  
    VESSEL3, 522  
SetPanelScaling  
    VESSEL3, 523  
SetPen  
    oapi::Sketchpad, 338  
SetPitchMomentScale  
    VESSEL, 417  
SetPMI  
    VESSEL, 385  
SetPosition  
    LightEmitter, 289  
    oapi::ScreenAnnotation, 334  
SetPositionRef  
    LightEmitter, 289  
SetPropellantEfficiency  
    VESSEL, 427  
SetPropellantMass  
    VESSEL, 426  
SetPropellantMaxMass  
    VESSEL, 425  
SetRange  
    GraphMFD, 278  
    PointLight, 331  
SetReentryTexture  
    VESSEL, 490  
SetRotationMatrix  
    VESSEL, 474  
SetRotDrag  
    VESSEL, 417  
SetSize  
    oapi::ScreenAnnotation, 334  
    VESSEL, 379  
SetSurfaceFrictionCoeff  
    VESSEL, 383  
SetText  
    oapi::ScreenAnnotation, 334  
SetTextAlign  
    oapi::Sketchpad, 339  
SetTextColor  
    oapi::Sketchpad, 339  
SetThrusterDir  
    VESSEL, 433  
SetThrusterGroupLevel  
    VESSEL, 445  
SetThrusterIsp  
    VESSEL, 437  
SetThrusterLevel  
    VESSEL, 438  
SetThrusterLevel\_SingleStep  
    VESSEL, 438  
SetThrusterMax0  
    VESSEL, 434  
SetThrusterRef  
    VESSEL, 432  
SetThrusterResource  
    VESSEL, 432  
SetTouchdownPoints  
    VESSEL, 382  
SetTransponderChannel  
    VESSEL, 454  
SetTrimScale  
    VESSEL, 419  
SetVariableDir  
    oapi::ParticleStream, 325  
SetVariablePos  
    oapi::ParticleStream, 325  
SetVessel  
    ExternMFD, 232  
SetVisibilityLimit  
    VESSEL, 379  
SetWheelbrakeLevel  
    VESSEL, 494  
SetWingAspect  
    VESSEL, 415  
SetWingEffectiveness  
    VESSEL, 416  
SetYawMomentScale

VESSEL, 418  
ShiftCentreOfMass  
    VESSEL, 472  
ShiftCG  
    VESSEL, 473  
ShiftExplicitPosition  
    LightEmitter, 290  
ShiftMesh  
    VESSEL, 463  
ShiftMeshes  
    VESSEL, 463  
SidRotPeriod  
    CELBODY2, 223  
Sketchpad  
    oapi::Sketchpad, 338  
Some useful general constants, 28  
SpotLight, 347  
    GetPenumbra, 350  
    GetUmbra, 350  
    SetAperture, 350  
    SpotLight, 349  
status  
    VESSELSTATUS, 532  
    VESSELSTATUS2, 536  
StoreStatus  
    MFD, 304  
Structure definitions, 29  
Style  
    oapi::Font, 235  
surf\_hdg  
    VESSELSTATUS2, 537  
surf\_lat  
    VESSELSTATUS2, 536  
surf\_lng  
    VESSELSTATUS2, 536  
Surface  
    oapiBeginBltGroup, 152  
    oapiBlt, 152, 153  
    oapiClearSurfaceColourKey, 153  
    oapiColourFill, 154  
    oapiCreateSurface, 154, 155  
    oapiCreateTextureSurface, 155  
    oapiDestroySurface, 155  
    oapiEndBltGroup, 156  
    oapiSetSurfaceColourKey, 156  
Surface and texture attributes, 41  
Surface base interface, 109  
Surface functions, 151  
surfId  
    RENDERTGT\_MAINWINDOW, 31  
TAlign\_horizontal  
    oapi::Sketchpad, 337  
TAlign\_vertical

oapi::Sketchpad, 337  
Text  
    oapi::Sketchpad, 341  
TextBox  
    oapi::Sketchpad, 342  
TexturePath  
    oapi::GraphicsClient, 254  
THGROUP\_ATT\_BACK  
    thrusterparam, 46  
THGROUP\_ATT\_BANKLEFT  
    thrusterparam, 46  
THGROUP\_ATT\_BANKRIGHT  
    thrusterparam, 46  
THGROUP\_ATT\_DOWN  
    thrusterparam, 46  
THGROUP\_ATT\_FORWARD  
    thrusterparam, 46  
THGROUP\_ATT\_LEFT  
    thrusterparam, 46  
THGROUP\_ATT\_PITCHDOWN  
    thrusterparam, 46  
THGROUP\_ATT\_PITCHUP  
    thrusterparam, 46  
THGROUP\_ATT\_RIGHT  
    thrusterparam, 46  
THGROUP\_ATT\_UP  
    thrusterparam, 46  
THGROUP\_ATT\_YAWLEFT  
    thrusterparam, 46  
THGROUP\_ATT\_YAWRIGHT  
    thrusterparam, 46  
THGROUP\_HOVER  
    thrusterparam, 46  
THGROUP\_MAIN  
    thrusterparam, 46  
THGROUP\_RETRO  
    thrusterparam, 46  
THGROUP\_USER  
    thrusterparam, 46  
THGROUP\_TYPE  
    thrusterparam, 46  
Thruster and thruster-group parameters, 45  
ThrusterGroupDefined  
    VESSEL, 444  
thrusterparam  
    ENGINE\_ATTITUDE, 46  
    ENGINE\_HOVER, 46  
    ENGINE\_MAIN, 45  
    ENGINE\_RETRO, 45  
    ENGINETYPE, 45  
    THGROUP\_ATT\_BACK, 46  
    THGROUP\_ATT\_BANKLEFT, 46  
    THGROUP\_ATT\_BANKRIGHT, 46  
    THGROUP\_ATT\_DOWN, 46

THGROUP\_ATT\_FORWARD, 46  
THGROUP\_ATT\_LEFT, 46  
THGROUP\_ATT\_PITCHDOWN, 46  
THGROUP\_ATT\_PITCHUP, 46  
THGROUP\_ATT\_RIGHT, 46  
THGROUP\_ATT\_UP, 46  
THGROUP\_ATT\_YAWLEFT, 46  
THGROUP\_ATT\_YAWRIGHT, 46  
THGROUP\_HOVER, 46  
THGROUP\_MAIN, 46  
THGROUP\_RETRO, 46  
THGROUP\_USER, 46  
THGROUP\_TYPE, 46  
Time functions, 112  
Title  
    MFD, 300  
    MFD2, 307  
tmul  
    vec, 40  
ToggleAttitudeMode  
    VESSEL, 449  
ToggleNavmode  
    VESSEL, 394  
TOP  
    oapi::Sketchpad, 337  
Top-level module callback functions, 199  
  
UNDERLINE  
    oapi::Font, 235  
Undock  
    VESSEL, 481  
unit  
    vec, 40  
UnregisterAnimation  
    VESSEL, 468  
UnregisterVisObject  
    oapi::GraphicsClient, 248  
Update  
    MFD, 300  
    MFD2, 307  
User input functions, 180  
UserInput  
    oapiOpenInputBox, 180  
Utility  
    oapiGetColour, 179  
    oapiRand, 179  
Utility functions, 179  
  
vdata  
    VESSELSTATUS, 532  
vec  
    \_M, 33  
    \_V, 34  
    crossp, 34  
    dist, 34  
    dotp, 34  
    length, 35  
    mul, 35  
    normalise, 35  
    operator\*, 36  
    operator\*=, 36  
    operator+, 37  
    operator+=, 37  
    operator-, 37, 38  
    operator-=, 38  
    operator/, 38, 39  
    operator/=, 39  
    outerp, 39  
    tmul, 40  
    unit, 40  
    veccpy, 40  
veccpy  
    vec, 40  
VECTOR3, 350  
Vectors and matrices, 31  
Version  
    CELBODY, 217  
    oapi::ModuleNV, 316  
    VESSEL, 376  
VESSEL, 351  
    ActivateNavmode, 394  
    AddAnimationComponent, 469  
    AddBeacon, 494  
    AddExhaust, 486–488  
    AddExhaustStream, 491, 492  
    AddForce, 423  
    AddMesh, 460, 461  
    AddParticleStream, 491  
    AddPointLight, 496  
    AddReentryStream, 492  
    AddSpotLight, 496  
    AttachChild, 485  
    AttachmentCount, 484  
    ClearAirfoilDefinitions, 409  
    ClearAttachments, 483  
    ClearBeacons, 495  
    ClearControlSurfaceDefinitions, 412  
    ClearDockDefinitions, 478  
    ClearLightEmitters, 498  
    ClearMeshes, 460, 501  
    ClearPropellantResources, 424  
    ClearThrusterDefinitions, 431  
    ClearVariableDragElements, 414  
    CopyMeshFromTemplate, 466  
    Create, 503  
    CreateAirfoil, 405  
    CreateAirfoil2, 406  
    CreateAirfoil3, 407

CreateAnimation, 468  
CreateAttachment, 482  
CreateControlSurface, 409  
CreateControlSurface2, 410  
CreateControlSurface3, 411  
CreateDock, 477  
CreatePropellantResource, 423  
CreateThruster, 429  
CreateThrusterGroup, 440  
CreateVariableDragElement, 413  
DeactivateNavmode, 394  
DefSetState, 387  
DefSetStateEx, 387  
DelAirfoil, 409  
DelAnimation, 469  
DelAnimationComponent, 470  
DelAttachment, 482  
DelBeacon, 495  
DelControlSurface, 411  
DelDock, 477  
DelExhaust, 489  
DelExhaustStream, 493  
DelLightEmitter, 497  
DelMesh, 462  
DelPropellantResource, 424  
DelThruster, 430  
DelThrusterGroup, 441, 499  
DetachChild, 486  
Dock, 480  
DockCount, 479  
DockingStatus, 480  
EditAirfoil, 408  
EnableIDS, 454  
EnableTransponder, 453  
GetADCtrlMode, 393  
GetAirfoilParam, 407  
GetAirspeed, 404  
GetAirspeedVector, 404  
GetAltitude, 399  
GetAngularAcc, 390  
GetAngularMoment, 390  
GetAngularVel, 389  
GetAnimPtr, 471  
GetAOA, 405  
GetApDist, 399  
GetArgPer, 398  
GetAtmDensity, 401  
GetAtmPressure, 402  
GetAtmRef, 401  
GetAtmTemperature, 401  
GetAttachmentHandle, 485  
GetAttachmentId, 484  
GetAttachmentIndex, 485  
GetAttachmentParams, 483  
GetAttachmentStatus, 484  
GetAttitudeLinLevel, 450  
GetAttitudeMode, 448  
GetAttitudeRotLevel, 449  
GetBank, 400  
GetBankMomentScale, 500  
GetBeacon, 496  
GetCameraDefaultDirection, 457  
GetCameraOffset, 456  
GetClassName, 377  
GetClipRadius, 380  
GetCOG\_elev, 382  
GetControlSurfaceLevel, 413  
GetCrossSections, 383  
GetCW, 414  
GetDamageModel, 377  
GetDefaultPropellantResource, 428  
GetDevMesh, 465  
GetDockHandle, 479  
GetDockParams, 479  
GetDockStatus, 479  
GetDrag, 420  
GetDragVector, 421  
GetDynPressure, 402  
GetEditorModule, 376  
GetElements, 395, 396  
GetEmptyMass, 381  
GetEnableFocus, 378  
GetEquPos, 400  
GetExhaustCount, 489  
GetExhaustLevel, 490  
GetExhaustSpec, 489, 490  
GetFlightModel, 377  
GetFlightStatus, 387  
GetForceVector, 422  
GetFuelMass, 429  
GetFuelRate, 429  
GetGlobalOrientation, 391  
GetGlobalPos, 388  
GetGlobalVel, 388  
GetGravityGradientDamping, 385  
GetGravityRef, 395  
GetGroundspeed, 403  
GetGroundspeedVector, 403  
GetGroupThruster, 443, 444  
GetGroupThrusterCount, 443  
GetHandle, 376  
GetHorizonAirspeedVector, 405  
GetIDS, 455  
GetISP, 439  
GetLift, 420  
GetLiftVector, 421  
GetLightEmitter, 497  
GetLinearMoment, 390

GetMachNumber, 402  
GetManualControlLevel, 447  
GetMass, 388  
GetMaxFuelMass, 428  
GetMesh, 464  
GetMeshCount, 464  
GetMeshName, 465  
GetMeshOffset, 464  
GetMeshTemplate, 465  
GetMeshVisibilityMode, 466  
GetName, 376  
GetNavChannel, 453  
GetNavController, 452  
GetNavmodeState, 395  
GetNavRecv, 501  
GetNavRecvFreq, 453  
GetNavSource, 456  
GetNosewheelSteering, 493  
GetPeDist, 398  
GetPitch, 399  
GetPitchMomentScale, 417  
GetPMI, 384  
GetPropellantCount, 424  
GetPropellantEfficiency, 426  
GetPropellantFlowrate, 427  
GetPropellantHandleByIndex, 424  
GetPropellantMass, 425  
GetPropellantMaxMass, 425  
GetRelativePos, 389  
GetRelativeVel, 389  
GetRotationMatrix, 474  
GetRotDrag, 416  
GetShipAirspeedVector, 405  
GetSize, 378  
GetSlipAngle, 405  
GetSMi, 397  
GetStatus, 386  
GetStatusEx, 386  
GetSuperstructureCG, 473  
GetSurfaceRef, 399  
GetThrusterCount, 431  
GetThrusterDir, 433  
GetThrusterGroupHandle, 442  
GetThrusterGroupLevel, 447  
GetThrusterHandleByIndex, 431  
GetThrusterIsp, 436  
GetThrusterIsp0, 435  
GetThrusterLevel, 437  
GetThrusterMax, 434, 435  
GetThrusterMax0, 433  
GetThrusterMoment, 439  
GetThrusterRef, 432  
GetThrusterResource, 431  
GetThrustVector, 421  
GetTorqueVector, 422  
GetTotalPropellantFlowrate, 427  
GetTotalPropellantMass, 426  
GetTouchdownPoints, 382  
GetTransponder, 455  
GetTrimScale, 418  
 GetUserThrusterGroupCount, 444  
 GetUserThrusterGroupHandleByIndex, 442  
GetWeightVector, 420  
GetWheelbrakeLevel, 494  
GetWingAspect, 415  
GetWingEffectiveness, 416  
GetYaw, 400  
GetYawMomentScale, 418  
Global2Local, 476  
GlobalRot, 474  
GroundContact, 392  
HorizonInvRot, 475  
HorizonRot, 475  
IncEngineLevel, 499  
IncThrusterGroupLevel, 445  
IncThrusterGroupLevel\_SingleStep, 446  
IncThrusterLevel, 438  
IncThrusterLevel\_SingleStep, 439  
InitNavRadios, 452  
InsertMesh, 461, 462  
LightEmitterCount, 497  
Local2Global, 475  
Local2Rel, 476  
MeshgroupTransform, 467  
MeshModified, 467  
NonsphericalGravityEnabled, 392  
OrbitStabilised, 392  
ParseScenarioLine, 503  
ParseScenarioLineEx, 498  
Playback, 472  
RecordEvent, 472  
Recording, 471  
RegisterAnimation, 468  
SaveDefaultState, 502  
SendBufferedKey, 451  
SetADCctrlMode, 393  
SetAlbedoRGB, 380  
SetAngularVel, 391  
SetAnimation, 471  
SetAttachmentParams, 483  
SetAttitudeLinLevel, 451  
SetAttitudeMode, 448  
SetAttitudeRotLevel, 450  
SetBankMomentScale, 500  
SetCameraCatchAngle, 458  
SetCameraDefaultDirection, 457  
SetCameraMovement, 459  
SetCameraOffset, 456

SetCameraRotationRange, 458  
SetCameraShiftRange, 459  
SetClipRadius, 381  
SetCOG\_elev, 501  
SetControlSurfaceLevel, 412  
SetCrossSections, 384  
SetCW, 414  
SetDefaultPropellantResource, 428  
SetDockMode, 481  
SetDockParams, 478  
SetElements, 397  
SetEmptyMass, 381  
SetEnableFocus, 378  
SetEngineLevel, 498  
SetExhaustScales, 499  
SetFuelMass, 429  
SetGlobalOrientation, 391  
SetGravityGradientDamping, 385  
SetIDSChannel, 454  
SetISP, 440  
SetLiftCoeffFunc, 419  
SetMaxFuelMass, 428  
SetMaxWheelbrakeForce, 494  
SetMeshVisibilityMode, 466  
SetMeshVisibleInternal, 502  
SetNavChannel, 452  
SetNavRecv, 501  
SetNosewheelSteering, 493  
SetPitchMomentScale, 417  
SetPMI, 385  
SetPropellantEfficiency, 427  
SetPropellantMass, 426  
SetPropellantMaxMass, 425  
SetReentryTexture, 490  
SetRotationMatrix, 474  
SetRotDrag, 417  
SetSize, 379  
SetSurfaceFrictionCoeff, 383  
SetThrusterDir, 433  
SetThrusterGroupLevel, 445  
SetThrusterIsp, 437  
SetThrusterLevel, 438  
SetThrusterLevel\_SingleStep, 438  
SetThrusterMax0, 434  
SetThrusterRef, 432  
SetThrusterResource, 432  
SetTouchdownPoints, 382  
SetTransponderChannel, 454  
SetTrimScale, 419  
SetVisibilityLimit, 379  
SetWheelbrakeLevel, 494  
SetWingAspect, 415  
SetWingEffectiveness, 416  
SetYawMomentScale, 418  
ShiftCentreOfMass, 472  
ShiftCG, 473  
ShiftMesh, 463  
ShiftMeshes, 463  
ThrusterGroupDefined, 444  
ToggleAttitudeMode, 449  
ToggleNavmode, 394  
Undock, 481  
UnregisterAnimation, 468  
Version, 376  
VESSEL, 375  
Vessel creation and destruction, 73  
Vessel functions, 77  
Vessel mesh visibility flags, 56  
Vessel module callback functions, 200  
VESSEL2, 503  
  clbkADCctrlMode, 512  
  clbkAnimate, 514  
  clbkConsumeBufferedKey, 515  
  clbkConsumeDirectKey, 515  
  clbkDockEvent, 514  
  clbkDrawHUD, 511  
  clbkFocusChanged, 508  
  clbkHUDMode, 513  
  clbkLoadGenericCockpit, 516  
  clbkLoadPanel, 516  
  clbkLoadStateEx, 507  
  clbkLoadVC, 518  
  clbkMFDMode, 513  
  clbkNavMode, 514  
  clbkPanelMouseEvent, 517  
  clbkPanelRedrawEvent, 517  
  clbkPlaybackEvent, 510  
  clbkPostCreation, 508  
  clbkPostStep, 509  
  clbkPreStep, 509  
  clbkRCSMode, 512  
  clbkSaveState, 507  
  clbkSetClassCaps, 506  
  clbkSetStateEx, 508  
  clbkVCMouseEvent, 519  
  clbkVCRedrawEvent, 519  
  clbkVisualCreated, 510  
  clbkVisualDestroyed, 511  
  VESSEL2, 506  
VESSEL3, 520  
  clbkDrawHUD, 527  
  clbkGeneric, 526  
  clbkGetRadiationForce, 529  
  clbkLoadPanel2D, 527  
  clbkPanelMouseEvent, 525  
  clbkPanelRedrawEvent, 526  
  clbkRenderHUD, 528  
  RegisterPanelArea, 524

RegisterPanelMFDGeometry, [523](#)  
SetPanelBackground, [522](#)  
SetPanelScaling, [523](#)  
VESSEL3, [522](#)  
VESSEL4, [530](#)  
    RegisterPanelArea, [530](#)  
    VESSEL4, [530](#)  
vessel\_clbk  
    ovcExit, [201](#)  
    ovcInit, [201](#)  
VesselCreation  
    oapiCreateVessel, [73](#)  
    oapiCreateVesselEx, [73](#)  
    oapiDeleteVessel, [74](#)  
VESSELSTATUS, [531](#)  
    flag, [533](#)  
    status, [532](#)  
    vdata, [532](#)  
VESSELSTATUS2, [533](#)  
    arot, [536](#)  
    flag, [535](#)  
    status, [536](#)  
    surf\_hdg, [537](#)  
    surf\_lat, [536](#)  
    surf\_lng, [536](#)  
VESSELSTATUS2::DOCKINFOSPEC, [537](#)  
VESSELSTATUS2::FUELSPEC, [537](#)  
VESSELSTATUS2::THRUSTSPEC, [538](#)  
Virtual cockpit functions, [159](#)  
VirtualCockpit  
    oapiVCRegisterArea, [159](#)  
    oapiVCRegisterHUD, [160](#)  
    oapiVCRegisterMFD, [160](#)  
    oapiVCSetAreaClickmode\_Quadrilateral, [161](#)  
    oapiVCSetAreaClickmode\_Spherical, [161](#)  
    oapiVCSetNeighbours, [162](#)  
    oapiVCTriggerRedrawArea, [162](#)  
Visual and mesh functions, [122](#)  
  
WriteStatus  
    MFD, [304](#)