

Orbiter API Reference Manual

2010 Edition

Generated by Doxygen 1.5.3

Sat Aug 21 03:42:42 2010

Contents

1	Orbiter API Reference Manual	1
2	Orbiter API Module Index	2
3	Orbiter API Hierarchical Index	4
4	Orbiter API Class Index	6
5	Orbiter API File Index	9
6	Orbiter API Page Index	9
7	Orbiter API Module Documentation	10
8	Orbiter API Class Documentation	190
9	Orbiter API File Documentation	516
10	Orbiter API Example Documentation	567
11	VESSEL2.cpp	568
12	Orbiter API Page Documentation	568

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 Orbiter API Module Index

2.1 Orbiter API Modules

Here is a list of all modules:

Configuration parameter identifiers	10
Render parameter identifiers	14
Bit flags for planetarium mode elements	14
Bit flags for blitting operations	15
Some useful general constants	16
Defines and Enumerations	17
Ephemeris data format bitflags	10
Handles	18
Mesh group editing flags	29
Bitflags for EXHAUSTSPEC flags field.	31
Light beacon shape parameters	31
Listentryflag	32
Listclbkflag	32
Animation flags	32
Thruster and thruster-group parameters	32
Airfoil orientation	34
Aerodynamic control surface types	34

Control surface axis orientation	35
Identifiers for visual events	35
Navigation mode identifiers	36
Manual control mode identifiers	37
Manual control device identifiers	37
RCS mode identifiers	38
HUD mode identifiers	38
MFD mode identifiers	39
MFD identifiers	40
Panel neighbour identifiers	40
Panel redraw event identifiers	41
Mouse event identifiers	41
Generic vessel message identifiers	42
Vessel mesh visibility flags	43
Navigation radio transmitter types	43
Object parameter flags	44
Keyboard key identifiers	170
Logical key ids	176
Structure definitions	17
Vectors and matrices	20
Local lighting interface	31
Orbiter API interface methods	44
Object access functions	51
Vessel creation and destruction	60
Body functions	61
Vessel functions	64
Coordinate transformations	79
Camera functions	83
Functions for planetary bodies	88

Surface base interface	95
Time functions	98
Navigation radio transmitter functions	103
Script interpreter functions	107
Visual and mesh functions	109
HUD, MFD and panel functions	120
Drawing support functions	133
Surface functions	138
Custom MFD mode definition	142
Virtual cockpit functions	145
Customisation - custom menu, dialogs	149
File IO Functions	156
Utility functions	164
User input functions	165
Onscreen annotations	166
Obsolete functions	168
Top-level module callback functions	183
General module callback functions	183
Vessel module callback functions	184
Plugin module callback functions	186

3 Orbiter API Hierarchical Index

3.1 Orbiter API Class Hierarchy

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

ANIMATION	190
ANIMATIONCOMP	191
ATMCONST	192
ATMOSPHERE	193
ATMOSPHERE::PRM_IN	196

ATMOSPHERE::PRM_OUT	196
ATMPARAM	197
BEACONLIGHTSPEC	197
CELBODY	200
CELBODY2	204
COLOUR4	209
oapi::DrawingTool	210
oapi::Brush	199
oapi::Font	218
oapi::Pen	309
ELEMENTS	210
ENGINESTATUS	211
EXHAUSTSPEC	212
ExternMFD	213
FogParam	217
oapi::GraphicsClient::LABELLIST	256
oapi::GraphicsClient::VIDEODATA	257
GROUPEDITSPEC	262
HELPCONTEXT	263
HUDPARAM	263
oapi::IVECTOR2	264
LaunchpadItem	264
LightEmitter	267
PointLight	310
SpotLight	328
LISTENTRY	273
MATERIAL	274
MATRIX3	274
MESHGROUP	275

MESHGROUP_TRANSFORM	276
MESHGROUPEX	277
MFD	278
GraphMFD	258
MFD2	286
oapi::ModuleNV	296
oapi::Module	291
oapi::GraphicsClient	220
NAVDATA	299
NTVERTEX	300
ORBITPARAM	301
oapi::ParticleStream	302
PARTICLESTREAMSPEC	307
oapi::ScreenAnnotation	313
oapi::Sketchpad	316
VECTOR3	331
VESSEL	332
VESSEL2	482
VESSEL3	499
VESSELSTATUS	508
VESSELSTATUS2	510
VESSELSTATUS2::DOCKINFOSPEC	514
VESSELSTATUS2::FUELSPEC	515
VESSELSTATUS2::THRUSTSPEC	515

4 Orbiter API Class Index

4.1 Orbiter API Class List

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

ANIMATION (Animation definition)	190
--	------------

ANIMATIONCOMP (Animation component definition)	191
ATMCONST (Planetary atmospheric constants structure)	192
ATMOSPHERE (Defines the physical atmospheric properties for a celestial body)	193
ATMOSPHERE::PRM_IN (Input parameters for atmospheric data calculation)	196
ATMOSPHERE::PRM_OUT (Output parameters for atmospheric data calculation)	196
ATMPARAM (Atmospheric parameters structure)	197
BEACONLIGHTSPEC (Vessel beacon light parameters)	197
oapi::Brush (A brush is a drawing resource for filling closed figures (rectangles, ellipses, polygons))	199
CELBODY (This is the base class for celestial body classes)	200
CELBODY2 (Extension to CELBODY class)	204
COLOUR4 (Colour definition)	209
oapi::DrawingTool (Base class for various 2-D drawing resources (fonts, pens, brushes, etc.))	210
ELEMENTS (Kepler orbital elements)	210
ENGINESTATUS (Engine status)	211
EXHAUSTSPEC (Engine exhaust render parameters)	212
ExternMFD (ExternMFD provides support for defining an MFD display in a plugin module)	213
FogParam (Distance fog render parameters)	217
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)	218
oapi::GraphicsClient (Base class for external graphics client modules)	220
oapi::GraphicsClient::LABELLIST (Label list description for celestial and surface markers)	256
oapi::GraphicsClient::VIDEODATA (Structure containing default video options, as stored in Orbiter.cfg)	257
GraphMFD (This class is derived from MFD and provides a template for MFD modes containing 2D graphs)	258
GROUPEDITSPEC (Structure used by oapiEditMeshGroup to define the group elements to be replaced)	262
HELPCONTEXT (Context information for an Orbiter ingame help page)	263
HUDPARAM (Mode-specific parameters for HUD mode settings)	263
oapi::IVECTOR2 (Integer-valued 2-D vector type)	264

LaunchpadItem (Base class to define launchpad items)	264
LightEmitter (Base class for defining a light source that can illuminate other objects)	267
LISTENTRY (Entry specification for selection list entry)	273
MATERIAL (Material definition)	274
MATRIX3 (3x3-element matrix)	274
MESHGROUP (Defines a mesh group (subset of a mesh))	275
MESHGROUP_TRANSFORM (This structure defines an affine mesh group transform (translation, rotation or scaling))	276
MESHGROUPEX (Extended mesh group definition)	277
MFD (This class acts as an interface for user defined MFD (multi functional display) modes)	278
MFD2 (Extended MFD class)	286
oapi::Module (Generic Orbiter plugin interface class)	291
oapi::ModuleNV (Generic Orbiter plugin interface class)	296
NAVDATA (Navigation transmitter data)	299
NTVERTEX (Vertex definition including normals and texture coordinates)	300
ORBITPARAM (Secondary orbital parameters derived from the primary ELEMENTS)	301
oapi::ParticleStream (Defines an array of "particles" (e.g. for exhaust and reentry effects, gas venting, condensation, etc.))	302
PARTICLESTREAMSPEC (Particle stream parameters)	307
oapi::Pen (A pen is a resource used for drawing lines and the outlines of closed figures such as rectangles, ellipses and polygons)	309
PointLight (Class for isotropic point light source)	310
oapi::ScreenAnnotation (Defines a block of text displayed on top of the simulation render window)	313
oapi::Sketchpad (A Sketchpad object defines an environment for drawing onto 2-D surfaces)	316
SpotLight (Class for directed spot light sources)	328
VECTOR3 (3-element vector)	331
VESSEL (Base class for objects of vessel type (spacecraft and similar))	332
VESSEL2 (Callback extensions to the VESSEL class)	482
VESSEL3 (Callback extensions to the VESSEL class)	499
VESSELSTATUS (Vessel status parameters (version 1))	508

VESSELSTATUS2 (Vessel status parameters (version 2))	510
VESSELSTATUS2::DOCKINFOSPEC (Dock info list)	514
VESSELSTATUS2::FUELSPEC (Propellant list)	515
VESSELSTATUS2::THRUSTSPEC (Thruster definition list)	515

5 Orbiter API File Index

5.1 Orbiter API File List

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

Orbitersdk/include/CelBodyAPI.h (Contains interface classes for celestial bodies: CELBODY and CELBODY2)	516
Orbitersdk/include/DrawAPI.h (2-D surface drawing support interface)	517
Orbitersdk/include/MFDAPI.h (Class interfaces for MFD instruments and MFD modes)	518
Orbitersdk/include/OrbiterAPI.h (General API interface functions)	518
Orbitersdk/include/VesselAPI.h (Contains the class interfaces for vessel objects (VESSEL , VESSEL2 , VESSEL3))	566

6 Orbiter API Page Index

6.1 Orbiter API Related Pages

Here is a list of all related documentation pages:

Planet Modules	568
Graphics Client Development	571
Deleted and obsolete functions and methods	571
Basics of orbital mechanics	571
Particle Streams HowTo	575
Vessel module concepts	576
Todo List	577
Deprecated List	578
Bug List	580

7 Orbiter API Module Documentation

7.1 Ephemeris data format bitflags

7.1.1 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

7.2 Configuration parameter identifiers

7.2.1 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_SURFACEPATCHAP** 0x0005
- #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_SURFACESPECULAR 0x0012
- #define CFGPRM_CSPHERETEXTURE 0x0013
- #define CFGPRM_CSPHEREINTENS 0x0014
- #define CFGPRM_LOCALLIGHT 0x0015
- #define CFGPRM_MAXLIGHT 0x0016

7.2.2 Define Documentation

7.2.2.1 #define CFGPRM_AMBIENTLEVEL 0x000E

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

Parameter type:

DWORD

7.2.2.2 #define CFGPRM_ATMFOG 0x0008

Flag for rendering distance fog within planetary atmospheres

Parameter type:

bool

7.2.2.3 #define CFGPRM_ATMHAZE 0x0007

Flag for rendering "atmospheric haze" over planets with atmospheres

Parameter type:

bool

7.2.2.4 #define CFGPRM_CLOUDS 0x0009

Flag for rendering planetary cloud layers

Parameter type:

bool

7.2.2.5 #define CFGPRM_CLOUDSHADOWS 0x000A

Flag for rendering cloud shadows on the planet surface

Parameter type:

bool

7.2.2.6 #define CFGPRM_CSPHEREINTENS 0x0014

Flag for rendering intensity of celestial sphere background textures

Parameter type:

double

7.2.2.7 #define CFGPRM_CSPHERETEXTURE 0x0013

File path for celestial sphere background textures

Parameter type:

*char

7.2.2.8 #define CFGPRM_LOCALLIGHT 0x0015

Flag for enabling local light sources

Parameter type:

bool

7.2.2.9 #define CFGPRM_MAXLIGHT 0x0016

Max number of light sources

Parameter type:

int

7.2.2.10 #define CFGPRM_OBJECTSHADOWS 0x0010

Flag for rendering object shadows on the planet surface

Parameter type:

bool

7.2.2.11 #define CFGPRM_OBJECTSPECULAR 0x0011

Flag for enabling specular reflections from objects

Parameter type:

bool

7.2.2.12 #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

7.2.2.13 #define CFGPRM_STARRENDERPRM 0x000C

Parameters for rendering stars on the celestial sphere.

Parameter type:

struct StarRenderPrm

7.2.2.14 #define CFGPRM_SURFACELIGHTBRT 0x0006

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

Parameter type:

double

7.2.2.15 #define CFGPRM_SURFACELIGHTS 0x0004

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

Parameter type:

bool

7.2.2.16 #define CFGPRM_SURFACEMAXLEVEL 0x0001

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

Parameter type:

DWORD

7.2.2.17 #define CFGPRM_SURFACEPATCHAP 0x0005

Angular aperture fraction over which to use high resolution patches (0-1)

Parameter type:

double

7.2.2.18 #define CFGPRM_SURFACEREFLECT 0x0002

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

Parameter type:

bool

7.2.2.19 #define CFGPRM_SURFACERIPPLE 0x0003

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

Parameter type:

bool

7.2.2.20 #define CFGPRM_SURFACESPECULAR 0x0012

Flag for enabling specular reflections from planetary surfaces

Parameter type:

bool

7.2.2.21 #define CFGPRM_VESSELSHADOWS 0x000F

Flag for rendering vessel shadows on the planet surface

Parameter type:

bool

7.3 Render parameter identifiers

7.3.1 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

7.3.2 Define Documentation

7.3.2.1 #define RP_STENCILDEPTH 3

stencil buffer depth of the render target

7.4 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**)

7.5 Bit flags for blitting operations

Defines

- #define **BLT_SRCCOLORKEY** 0x1
Use source surface colour key for transparency.
- #define **BLT_TGTCOLORKEY** 0x2

7.5.1 Define Documentation

7.5.1.1 #define BLT_TGTCOLORKEY 0x2

Use target surface colour key for transparency

7.6 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

7.7 Defines and Enumerations

Modules

- Ephemeris data format bitflags
- Handles
- Mesh group editing flags
- Bitflags for EXHAUSTSPEC flags field.
- Light beacon shape parameters
- Listentryflag
- Listclbkflag
- Animation flags
- 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
- Generic vessel message identifiers
- Vessel mesh visibility flags
- Navigation radio transmitter types
- Object parameter flags
- Keyboard key identifiers
- Logical key ids

7.8 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.

7.9 Handles

Typedefs

- `typedef void * OBJHANDLE`
- `typedef void * VISHANDLE`
- `typedef void * MESHHANDLE`
- `typedef int * DEVMESHHANDLE`
- `typedef void * SURFHANDLE`
- `typedef void * PANELHANDLE`
- `typedef void * FILEHANDLE`
- `typedef void * INTERPRETERHANDLE`
- `typedef void * THRUSTER_HANDLE`
- `typedef void * THGROUP_HANDLE`
- `typedef void * PROPELLANT_HANDLE`
- `typedef void * PSTREAM_HANDLE`
- `typedef void * DOCKHANDLE`
- `typedef void * ATTACHMENTHANDLE`
- `typedef void * AIRFOILHANDLE`
- `typedef void * CTRLSURFHANDLE`
- `typedef void * NAVHANDLE`
- `typedef void * ANIMATIONCOMPONENT_HANDLE`
- `typedef void * LAUNCHPADITEM_HANDLE`
- `typedef void * NOTEHANDLE`

7.9.1 Typedef Documentation

7.9.1.1 `typedef void* AIRFOILHANDLE`

Handle for vessel airfoils

7.9.1.2 `typedef void* ANIMATIONCOMPONENT_HANDLE`

Handle for animation components

7.9.1.3 `typedef void* ATTACHMENTHANDLE`

Handle for vessel passive attachment points

7.9.1.4 `typedef void* CTRLSURFHANDLE`

Handle for vessel aerodynamic control surfaces

7.9.1.5 `typedef int* DEVMESSHHANDLE`

Handle for graphics-client-specific meshes

7.9.1.6 `typedef void* DOCKHANDLE`

Handle for vessel docking ports

7.9.1.7 `typedef void* FILEHANDLE`

Handle for file streams

Examples:

[clbkLoadStateEx.cpp](#).

7.9.1.8 `typedef void* INTERPRETERHANDLE`

Handle for script interpreters

7.9.1.9 `typedef void* LAUNCHPADITEM_HANDLE`

Handle for custom items added to Launchpad "Extra" list

7.9.1.10 `typedef void* MESHHANDLE`

Handle for meshes

7.9.1.11 `typedef void* NAVHANDLE`

Handle for a navigation radio transmitter (VOR, ILS, IDS, XPDR)

7.9.1.12 `typedef void* NOTEHANDLE`

Handle for onscreen annotation objects

7.9.1.13 `typedef void* OBJHANDLE`

Handle for objects (vessels, stations, planets)

Examples:

[VESSEL2.cpp](#).

7.9.1.14 `typedef void* PANELHANDLE`

Handle for 2D instrument panels

7.9.1.15 `typedef void* PROPELLANT_HANDLE`

Propellant resource handle

7.9.1.16 `typedef void* PSTREAM_HANDLE`

Handle for particle streams

7.9.1.17 `typedef void* SURFHANDLE`

Handle for bitmap surfaces and textures (panels and panel items)

7.9.1.18 `typedef void* THGROUP_HANDLE`

Handle for logical thruster groups

7.9.1.19 `typedef void* THRUSTER_HANDLE`

Handle for thrusters

7.9.1.20 `typedef void* VISHANDLE`

Handle for visuals

7.10 Vectors and matrices

7.10.1 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.

7.10.2 Function Documentation

7.10.2.1 **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:

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

7.10.2.2 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](#).

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

Vector (cross) product of two vectors.

Parameters:

$\leftarrow a$ First vector operand

$\leftarrow b$ Second vector operand

Returns:

Vector product $a \times b$

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

Distance between two points.

Parameters:

$\leftarrow a$ First point

$\leftarrow b$ Second point

Returns:

Distance between a and b

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

Scalar (inner, dot) product of two vectors.

Parameters:

$\leftarrow a$ First vector operand

$\leftarrow b$ Second vector operand

Returns:

Scalar product $a \cdot b$

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

Length (L2-norm) of a vector.

Parameters:

a Vector operand

Returns:

Vector norm $|a|_2$

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

Matrix-matrix multiplication.

Parameters:

$\leftarrow A$ First matrix operand

$\leftarrow B$ Second matrix operand

Returns:

Result of AB

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

Matrix-vector multiplication.

Parameters:

$\leftarrow A$ matrix operand

$\leftarrow b$ vector operand

Returns:

Result of Ab

7.10.2.9 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.

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

Product of matrix and scalar.

Parameters:

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

Returns:

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

7.10.2.11 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 $a*f$.

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

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

Parameters:

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

Returns:

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

7.10.2.13 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.

7.10.2.14 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*)

7.10.2.15 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*.

7.10.2.16 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.

7.10.2.17 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*)

7.10.2.18 VECTOR3 operator- (const VECTOR3 & a) [inline]

Vector unary minus -a.

Parameters:

$\leftarrow a$ Vector operand

Returns:

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

7.10.2.19 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.

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

Vector subtraction-assignment a -= b.

Parameters:

$\leftrightarrow a$ Left-hand vector operand

$\leftarrow b$ Right-hand vector operand

Returns:

Replaces a with a-b and returns the result.

7.10.2.21 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.

7.10.2.22 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.

7.10.2.23 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.

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

Vector-scalar division-assignment a /= f.

Parameters:

$\leftarrow a$ Left-hand vector operand

$\leftarrow f$ Right-hand scalar operand

Returns:

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

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

Outer product of two vectors.

Parameters:

$\leftarrow a$ First vector operand

$\leftarrow b$ Second vector operand

Returns:

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

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

Matrix transpose-vector multiplication.

Parameters:

- ← A matrix operand
- ← b vector operand

Returns:

Result of $A^T b$

7.10.2.27 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.

7.10.2.28 void vecpy (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

7.11 Mesh group editing flags

7.11.1 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.

7.12 Bitflags for EXHAUSTSPEC flags field.

7.12.1 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

7.13 Local lighting interface

7.13.1 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.

7.14 Light beacon shape parameters

7.14.1 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

7.15 Listentryflag

See also:

[LISTENTRY](#)

7.16 Listclbkflag

See also:

[LISTENTRY](#)

7.17 Animation flags

7.17.1 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

7.18 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.

7.18.1 Enumeration Type Documentation

7.18.1.1 enum **ENGINETYPE**

Thruster group identifiers (obsolete).

Enumerator:

- ENGINE_MAIN** main thrusters
- ENGINE_RETRO** retro thrusters
- ENGINE_HOVER** hover thrusters
- ENGINE_ATTITUDE** attitude (RCS) thrusters

7.18.1.2 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

7.19 Airfoil orientation

Enumerations

- enum [AIRFOIL_ORIENTATION](#) { [LIFT_VERTICAL](#), [LIFT_HORIZONTAL](#) }
Lift vector orientation for airfoils.

7.19.1 Enumeration Type Documentation

7.19.1.1 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)

7.20 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.

7.20.1 Enumeration Type Documentation

7.20.1.1 enum AIRCTRL_TYPE

Control surfaces provide attitude and drag control during atmospheric flight.

See also:

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

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

7.21 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

7.21.1 Define Documentation

7.21.1.1 #define AIRCTRL_AXIS_AUTO 0

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

See also:

[VESSEL::CreateControlSurface](#),

[VESSEL::CreateControlSurface2](#),

VES-

[SEL::CreateControlSurface3](#) automatic orientation

7.22 Identifiers for visual events

7.22.1 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_MESHOPS 7
Shift a mesh (context: mesh index).
- #define EVENT_VESSEL_MODMESHGROUP 8
A mesh group has been modified.

7.23 Navigation mode identifiers

7.23.1 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

7.24 Manual control mode identifiers

7.24.1 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

7.25 Manual control device identifiers

7.25.1 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

7.26 RCS mode identifiers

7.26.1 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.

7.27 HUD mode identifiers

7.27.1 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.

7.28 MFD mode identifiers

7.28.1 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.

7.29 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.

7.30 Panel neighbour identifiers

7.30.1 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

7.31 Panel redraw event identifiers

7.31.1 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 0x00
Don't generate redraw events.
- #define PANEL_REDRAW_ALWAYS 0x01
Generate event at each frame.
- #define PANEL_REDRAW_MOUSE 0x02
Generate event on mouse event.
- #define PANEL_REDRAW_INIT 0x03
Initialisation event.
- #define PANEL_REDRAW_USER 0x04
User-generated event.

7.32 Mouse event identifiers

7.32.1 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_LBDOWN** 0x01
Left button down event.
- #define **PANEL_MOUSE_RBDOWN** 0x02
Right button down event.
- #define **PANEL_MOUSE_LBUP** 0x04
Left button release event.
- #define **PANEL_MOUSE_RBUP** 0x08
Right button release event.
- #define **PANEL_MOUSE_LBPRESSED** 0x10
Left button down (continuous).
- #define **PANEL_MOUSE_RBPRESSED** 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.

7.33 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

7.34 Vessel mesh visibility flags

7.34.1 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.

7.35 Navigation radio transmitter types

7.35.1 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

7.36 Object parameter flags

7.36.1 Detailed Description

Used by `oapiGetObjectParam()`

Defines

- `#define OBJPRM_PLANET_SURFACELEVEL 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).

7.37 Orbiter API interface methods

7.37.1 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.

7.37.2 Function Documentation

7.37.2.1 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);
```

7.37.2.2 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.

7.37.2.3 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.

7.37.2.4 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.

7.37.2.5 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](#)

7.37.2.6 OAPIFUNC HINSTANCE oapiGetOrbiterInstance ()

Returns the instance handle for the running Orbiter application.

Returns:

Orbiter instance handle

7.37.2.7 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](#)

7.37.2.8 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.

7.37.2.9 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.

7.37.2.10 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:

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

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](#)

7.37.2.11 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](#)

7.37.2.12 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

7.37.2.13 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.

7.37.2.14 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](#)

7.38 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.

7.38.1 Function Documentation

7.38.1.1 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](#)

7.38.1.2 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](#)

7.38.1.3 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 (≥ 0).

7.38.1.4 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.

7.38.1.5 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.

7.38.1.6 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](#)

7.38.1.7 OAPIFUNC OBJHANDLE oapiGetGbodyByIndex (int *index*)

Returns the handle of a celestial body (sun, planet or moon) identified 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](#)

7.38.1.8 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](#)

7.38.1.9 OAPIFUNC DWORD oapiGetGbodyCount ()

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

Returns:

Number of objects.

7.38.1.10 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](#)

7.38.1.11 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](#)

7.38.1.12 OAPIFUNC DWORD oapiGetObjectCount ()

Returns the number of objects currently present in the simulation.

Returns:

object count

See also:

[oapiGetObjectByIndex](#), [oapiGetObjectType](#)

7.38.1.13 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.

7.38.1.14 OAPIFUNC const void* oapiGetObjectParam (OBJHANDLE *hObj*, DWORD *paramtype*)

Returns an object-specific configuration parameter.

Parameters:

hObj object handle

paramtype 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](#)

7.38.1.15 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](#)

7.38.1.16 OAPIFUNC OBJHANDLE oapiGetVesselByIndex (int *index*)

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

Parameters:

index object index (0 <= index < [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](#)

7.38.1.17 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](#)

7.38.1.18 OAPIFUNC DWORD oapiGetVesselCount ()

Returns the number of vessels currently present in the simulation.

Returns:

Vessel count.

See also:

[oapiGetVesselByIndex](#)

7.38.1.19 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.

7.38.1.20 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](#)

7.38.1.21 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](#)

7.39 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.

7.39.1 Function Documentation

7.39.1.1 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](#)

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

Creates a new vessel via a VESSELSTATUSx (x >= 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](#)

7.39.1.3 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](#)

7.40 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.

7.40.1 Function Documentation

7.40.1.1 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](#)

7.40.1.2 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](#)

7.40.1.3 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](#)

7.40.1.4 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](#)

7.40.1.5 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](#)

7.40.1.6 OAPIFUNC double oapiGetSize (OBJHANDLE hObj)

Returns the size (mean radius) of an object.

Parameters:

hObj object handle

Returns:

Object size (mean radius) in meter.

7.41 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** [oapiGetAirspeed](#) (**OBJHANDLE** hVessel, double *airspeed)

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

- 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 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.

7.41.1 Function Documentation

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

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

Parameters:

hVessel vessel handle

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

Returns:

Error flag (*false* on failure)

Note:

This function works even for planets or moons without atmosphere. It returns an "airspeed-equivalent" value.

7.41.1.2 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.

Parameters:

hVessel vessel handle

speedvec pointer to variable receiving airspeed vector [m/s in x,y,z]

Returns:

Error flag (*false* on failure)

Note:

This function returns the airspeed vector with respect to the local horizon reference frame. To get the vector with respect to the local vessel coordinates, use [oapiGetShipAirspeedVector\(\)](#).

7.41.1.3 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.

7.41.1.4 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.

7.41.1.5 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](#)

7.41.1.6 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](#)

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

Returns a handle to a vessel docking port.

Parameters:

hVessel vessel handle

n docking port index (≥ 0)

Returns:

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

See also:

[VESSEL::GetDockHandle](#)

7.41.1.8 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](#)

7.41.1.9 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.

7.41.1.10 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;
```

7.41.1.11 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.

7.41.1.12 OAPIFUNC BOOL oapiGetFocusAirspeed (double * *airspeed*)

Returns the current focus vessel's airspeed w.r.t. the closest planet or moon.

Parameters:

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

Returns:

Error flag (*false* on failure)

See also:

[oapiGetFocusAirspeedVector](#)

7.41.1.13 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.

Parameters:

speedvec pointer to variable receiving airspeed vector [m/s in x,y,z]

Returns:

Error flag (*false* on failure)

See also:

[oapiGetFocusAirspeed](#)

7.41.1.14 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)

7.41.1.15 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)

7.41.1.16 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](#)

7.41.1.17 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](#)

7.41.1.18 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)

7.41.1.19 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](#)

7.41.1.20 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](#)

7.41.1.21 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](#)

7.41.1.22 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](#)

7.41.1.23 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](#)

7.41.1.24 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](#)

7.41.1.25 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.

Parameters:

speedvec pointer to variable receiving airspeed vector [m/s in x,y,z]

Returns:

Error flag (*false* on failure)

7.41.1.26 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](#)

7.41.1.27 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](#)

7.41.1.28 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.

7.41.1.29 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](#)

7.41.1.30 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* \geq # propellant resources

7.41.1.31 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](#)

7.41.1.32 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](#)

7.41.1.33 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.

Parameters:

hVessel vessel handle

speedvec pointer to variable receiving airspeed vector [m/s in x,y,z]

Returns:

Error flag (*false* on failure)

Note:

This function returns the airspeed vector with respect to the vessel's frame of reference. To get the vector with respect to the local horizon's frame of reference, use [oapiGetAirspeedVector\(\)](#).

7.41.1.34 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](#)

7.41.1.35 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.

7.41.1.36 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.

7.41.1.37 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](#)

7.41.1.38 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](#)

7.41.1.39 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](#)

7.42 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.

7.42.1 Function Documentation

7.42.1.1 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](#)

7.42.1.2 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](#)

7.42.1.3 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\(const MATRIX3&,const VECTOR3&\)](#)

7.42.1.4 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](#)

7.42.1.5 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

→ ***loc*** point mapped into local coordinates

Note:

This function maps global point *glob* into the local reference frame of body *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}_{hObj} , \vec{p}_{hObj} are the body's rotation matrix and global position, respectively.

See also:

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

7.42.1.6 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.

Parameters:

- ← ***hObj*** object handle
- ← ***loc*** point in cartesian coordinates of the local object frame [m]
- ***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](#), [oapiEquToGlobal](#), [oapiGlobalToEqu](#)

7.42.1.7 OAPIFUNC void oapiLocalToGlobal (OBJHANDLE *hObj*, const VECTOR3 * *loc*, VECTOR3 * *glob*)

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

Parameters:

- ← ***hObj*** object handle
- ← ***loc*** point in local coordinates of the object frame
- ***glob*** point mapped into global coordinates

Note:

This function maps point *loc* given in local coordinates of *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}_{hObj} , \vec{p}_{hObj} are the body's rotation matrix and global position, respectively.

See also:

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

7.42.1.8 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.

7.43 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.

7.43.1 Function Documentation

7.43.1.1 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.

7.43.1.2 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](#)

7.43.1.3 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.

7.43.1.4 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](#)

7.43.1.5 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](#)

7.43.1.6 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](#)

7.43.1.7 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](#)

7.43.1.8 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.

7.43.1.9 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.

7.43.1.10 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.

7.43.1.11 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.

7.43.1.12 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.

7.43.1.13 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).

7.43.1.14 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](#)

7.43.1.15 OAPIFUNC double oapiCameraTargetDist ()

Returns the distance between the camera and its target [m].

Returns:

Distance between camera and camera target [m].

7.43.1.16 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](#)

7.44 Functions for planetary bodies

7.44.1 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 siderial 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.

7.44.2 Function Documentation

7.44.2.1 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.

7.44.2.2 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](#)

7.44.2.3 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](#)

7.44.2.4 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](#)

7.44.2.5 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 ω is the rotation angle returned by [oapiGetPlanetCurrentRotation\(\)](#).

7.44.2.6 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 ϕ 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](#)

7.44.2.7 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 $\epsilon = 10^{-10}$

7.44.2.8 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](#)

7.44.2.9 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 θ is the longitude of the ascending node of the equator, as returned by [oapiGetPlanetTheta\(\)](#), and φ 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](#)

7.44.2.10 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](#)

7.44.2.11 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](#)

7.44.2.12 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.

7.44.2.13 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](#)

7.45 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.

7.45.1 Function Documentation

7.45.1.1 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.

7.45.1.2 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.

7.45.1.3 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.

7.45.1.4 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.

7.45.1.5 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\(\)](#))

$0 \leq \text{pad} < \text{oapiGetBasePadCount()}$ 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

7.45.1.6 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](#)

7.46 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.

7.46.1 Function Documentation

7.46.1.1 OAPIFUNC double [oapiGetFrameRate](#) ()

Returns current simulation frame rate (frames/sec).

Returns:

Current frame rate (fps)

7.46.1.2 OAPIFUNC bool [oapiGetPause](#) ()

Returns the current simulation pause state.

Returns:

true if simulation is currently paused, *false* if it is running.

See also:

[oapiSetPause](#)

7.46.1.3 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](#)

7.46.1.4 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.

7.46.1.5 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\(\)](#).

7.46.1.6 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](#)

7.46.1.7 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.

7.46.1.8 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](#)

7.46.1.9 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](#)

7.46.1.10 OAPIFUNC void oapiSetPause (bool *pause*)

Sets the simulation pause state.

Parameters:

pause *true* to pause the simulation, *false* to resume.

See also:

[oapiGetPause](#)

7.46.1.11 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](#)

7.46.1.12 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 [1,100000]. 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](#)**7.46.1.13 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.

7.47 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.

7.47.1 Function Documentation

7.47.1.1 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](#)

7.47.1.2 OAPIFUNC int oapiGetNavData (NAVHANDLE *hNav*, NAVDATA * *data*)

Returns information about a NAV transmitter.

Parameters:

← *hNav* NAV transmitter handle

→ *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](#)

7.47.1.3 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

7.47.1.4 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](#)

7.47.1.5 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](#)

7.47.1.6 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](#)

7.47.1.7 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](#)

7.47.1.8 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](#)

7.47.1.9 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.

7.48 Script interpreter functions

Functions

- OAPIFUNC INTERPRETERHANDLE [oapiCreateInterpreter](#) ()
Returns a handle to a new interpreter instance.
- OAPIFUNC int [oapiDelInterpreter](#) (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*)

7.48.1 Function Documentation

7.48.1.1 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](#)

7.48.1.2 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](#)

7.48.1.3 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](#)

7.48.1.4 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](#)

7.49 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.

7.49.1 Typedef Documentation

7.49.1.1 **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.

7.49.2 Function Documentation

7.49.2.1 **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](#)

7.49.2.2 **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.

7.49.2.3 OAPIFUNC bool oapiDeleteMaterial (MESHHANDLE *hMesh*, DWORD *idx*)

Delete a material definition from the mesh.

Parameters:

hMesh mesh handle

idx material index (≥ 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](#)

7.49.2.4 OAPIFUNC void oapiDeleteMesh (MESHHANDLE *hMesh*)

Removes a mesh from memory.

Parameters:

hMesh mesh handle

7.49.2.5 OAPIFUNC int oapiEditMeshGroup (MESHHANDLE *hMesh*, DWORD *grpidx*, GROU-PEDITSPEC * *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*)`

7.49.2.6 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.

7.49.2.7 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](#)

7.49.2.8 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.

7.49.2.9 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.

7.49.2.10 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.

7.49.2.11 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 oapiEditMeshGroup instead.

See also:

[oapiEditMeshGroup](#)

7.49.2.12 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.

7.49.2.13 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 ($>= 0$)

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](#)

7.49.2.14 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.

7.49.2.15 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](#)

7.49.2.16 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.

7.49.2.17 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.

7.49.2.18 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.

7.49.2.19 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 (≥ 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.

7.49.2.20 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\(MESHHANDLE,DWORD,DWORD\)](#)

7.49.2.21 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\(DEVMESSHHANDLE,DWORD,DWORD\)](#)

7.49.2.22 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.

7.50 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.

7.50.1 Function Documentation

7.50.1.1 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](#)

7.50.1.2 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).

7.50.1.3 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\(\)](#)

7.50.1.4 OAPIFUNC int oapiGetHUDMode ()

Query current HUD (head up display) mode.

Returns:

Current HUD mode

See also:

[HUD Modes](#), [oapiGetHUDMode\(const HUDPARAM*\)](#), [oapiSetHUDMode](#)

7.50.1.5 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](#)

7.50.1.6 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](#)

7.50.1.7 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 ($>=0$)

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](#)

7.50.1.8 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](#)

7.50.1.9 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 (≥ 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.

7.50.1.10 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](#)

7.50.1.11 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](#)

7.50.1.12 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](#)

7.50.1.13 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](#)

7.50.1.14 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_MOVOUT_{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](#)

7.50.1.15 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.

7.50.1.16 OAPIFUNC int oapiSendMFDKey (int *mfd*, DWORD *key*)

Sends a keystroke to an [MFD](#).

Parameters:

mfd 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](#)

7.50.1.17 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 [VES-SEL2::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

7.50.1.18 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)

7.50.1.19 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\(const HUDPARAM*\)](#)

7.50.1.20 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](#)

7.50.1.21 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](#)

7.50.1.22 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.

7.50.1.23 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](#)

7.50.1.24 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](#)

7.50.1.25 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](#)

7.50.1.26 OAPIFUNC void oapiTriggerPanelRedrawArea (int *panel_id*, int *area_id*)

Triggers a redraw notification for a panel area.

Parameters:

panel_id panel identifier ($>=0$)

area_id area identifier ($>=0$)

Note:

The redraw notification is ignored if the requested panel is not currently displayed.

See also:

[oapiTriggerRedrawArea](#)

7.50.1.27 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.

7.51 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.

7.51.1 Function Documentation

7.51.1.1 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](#)

7.51.1.2 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.

7.51.1.3 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](#)

7.51.1.4 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](#)

7.51.1.5 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](#)

7.51.1.6 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](#)

7.51.1.7 OAPIFUNC void oapiReleaseBrush (oapi::Brush * *brush*)

Release a brush resource.

Parameters:

brush pointer to brush resource

See also:

[oapiCreateBrush](#)

7.51.1.8 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](#)

7.51.1.9 OAPIFUNC void oapiReleaseFont (oapi::Font * *font*)

Release a font resource.

Parameters:

font pointer to font resource

See also:

[oapiCreateFont](#)

7.51.1.10 OAPIFUNC void oapiReleasePen (oapi::Pen * *pen*)

Release a pen resource.

Parameters:

pen pointer to pen resource

See also:

[oapiCreatePen](#)

7.51.1.11 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](#)

7.52 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 void oapiColourFill (SURFHANDLE tgt, DWORD fillcolor, int tgtx=0, int tgty=0, int w=0, int h=0)
Fill an area of the target surface with a uniform colour.

7.52.1 Function Documentation

7.52.1.1 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.

7.52.1.2 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.

Parameters:

- tgt* target surface
- src* source surface
- tgtx* left edge of target rectangle [pixel]
- tgyt* top edge of target rectangle [pixel]
- srcx* left edge of source rectangle [pixel]
- srcy* 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.

7.52.1.3 OAPIFUNC void oapiClearSurfaceColourKey (SURFHANDLE *surf*)

Clear a previously defined colour key.

Parameters:

surf surface handle

See also:

[oapiSetSurfaceColourKey](#), [oapiBlt](#)

7.52.1.4 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.

7.52.1.5 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](#)

7.52.1.6 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](#)

7.52.1.7 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.

7.52.1.8 OAPIFUNC void oapiDestroySurface (SURFHANDLE *surf*)

Destroy a surface previously created with oapiCreateSurface.

Parameters:

surf surface handle

7.52.1.9 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](#)

7.53 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.

7.53.1 Function Documentation

7.53.1.1 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](#)

7.53.1.2 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

7.53.1.3 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](#)

7.53.1.4 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).

7.54 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.

7.54.1 Function Documentation

7.54.1.1 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);
```

7.54.1.2 OAPIFUNC void oapiVCRegisterArea (int *id*, const RECT & *tgrect*, 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
- tgrect* 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.

7.54.1.3 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.

7.54.1.4 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](#)

7.54.1.5 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](#)

7.54.1.6 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](#)

7.54.1.7 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.

7.54.1.8 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.

7.55 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.

7.55.1 Function Documentation

7.55.1.1 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.

7.55.1.2 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\(\)](#).

7.55.1.3 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](#)

7.55.1.4 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.

7.55.1.5 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](#)

7.55.1.6 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\(\)](#).

7.55.1.7 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](#)

7.55.1.8 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](#)

7.55.1.9 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*.

7.55.1.10 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*.

7.55.1.11 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](#)

7.55.1.12 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](#)

7.55.1.13 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](#)

7.55.1.14 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](#)

7.56 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 [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.

7.56.1 Function Documentation

7.56.1.1 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.

7.56.1.2 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](#)

7.56.1.3 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.

7.56.1.4 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.

7.56.1.5 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.

7.56.1.6 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.

7.56.1.7 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.

7.56.1.8 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](#).

7.56.1.9 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 ("").

7.56.1.10 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

7.56.1.11 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

7.56.1.12 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

7.56.1.13 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](#)

7.56.1.14 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

7.56.1.15 OAPIFUNC void oapiWriteLine (FILEHANDLE *file*, char * *line*)

Writes a line to a file.

Parameters:

file file handle

line line to be written (zero-terminated)

7.56.1.16 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.

7.56.1.17 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

7.56.1.18 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

7.56.1.19 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)

7.56.1.20 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

7.57 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.

7.57.1 Function Documentation

7.57.1.1 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 `oapiSetSurface-ColourKey`. 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.

7.57.1.2 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.

7.58 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)

7.58.1 Function Documentation

7.58.1.1 OAPIFUNC void oapiOpenInputBox (char * *title*, bool(*)(void *, char *, void *) *Clbk*, char * *buf* = 0, int *vislen* = 20, void * *usrdata* = 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

usrdata 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](#)

7.59 Onscreen annotations

7.59.1 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.

7.59.2 Function Documentation

7.59.2.1 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](#)

7.59.2.2 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](#)

7.59.2.3 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](#)

7.59.2.4 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](#)

7.59.2.5 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](#)

7.59.2.6 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](#)

7.60 Obsolete functions

Functions

- OAPIFUNC OBJHANDLE oapiGetStationByName (char *name)
- OAPIFUNC OBJHANDLE oapiGetStationByIndex (int index)
- 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 DWORD [oapiGetStationCount](#) ()
- 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.

7.60.1 Function Documentation

7.60.1.1 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/m³]

Note:

Pressure and density are calculated using an exponential barometric equation, without accounting for local variations.

See also:

[oapiGetAtm](#)

7.60.1.2 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 [kg/m³]

Note:

Pressure and density are calculated using an exponential barometric equation, without accounting for local variations.

See also:

[oapiGetAtm](#)

7.60.1.3 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](#)

7.60.1.4 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.

7.60.1.5 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.

7.60.1.6 OAPIFUNC int oapiRegisterMFDMode (MFDMODESPEC & *spec*)

Register a custom MFD mode.

Deprecated

This function has been replaced by [oapiRegisterMFDMode\(MFDMODESPECEX&\)](#).

See also:

[oapiRegisterMFDMode\(MFDMODESPECEX&\)](#)

7.61 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
'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
'H' key
- #define **OAPI_KEY_J** 0x24
'J' key
- #define **OAPI_KEY_K** 0x25

- #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.

7.62 Logical key ids

Defines

- #define **OAPI_LKEY_CockpitRotateLeft** 0
rotate camera left in cockpit view
- #define **OAPI_LKEY_CockpitRotateRight** 1

- #define OAPI_LKEY_CockpitRotateUp 2
rotate camera right in cockpit view
- #define OAPI_LKEY_CockpitRotateDown 3
rotate camera up in cockpit view
- #define OAPI_LKEY_CockpitDontLean 4
rotate camera down in cockpit view
- #define OAPI_LKEY_CockpitLeanForward 5
return to default cockpit camera position
- #define OAPI_LKEY_CockpitLeanLeft 6
move cockpit camera forward
- #define OAPI_LKEY_CockpitLeanRight 7
move cockpit camera left
- #define OAPI_LKEY_CockpitResetCam 8
move cockpit camera right
- #define OAPI_LKEY_CockpitResetCam 8
rotate and shift cockpit camera back to default
- #define OAPI_LKEY_PanelShiftLeft 9
rotate and shift cockpit camera back to default
- #define OAPI_LKEY_PanelShiftRight 10
shift 2D instrument panel left
- #define OAPI_LKEY_PanelShiftUp 11
shift 2D instrument panel right
- #define OAPI_LKEY_PanelShiftDown 12
shift 2D instrument panel up
- #define OAPI_LKEY_PanelShiftUp 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_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
toggle on-screen info block on/off
- #define **OAPI_LKEY_ToggleFPS** 94
toggle frame rate display on/off
- #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

7.63 Top-level module callback functions

7.63.1 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

7.64 General module callback functions

7.64.1 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.

7.64.2 Function Documentation

7.64.2.1 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.

7.64.2.2 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](#).

7.65 Vessel module callback functions

7.65.1 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.

7.65.2 Function Documentation

7.65.2.1 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](#).

7.65.2.2 DLLCLBK VESSEL* `ovcInit (OBJHANDLE hvessel, int flightmodel)`

Vessel instance creation notification.

Parameters:

hessel 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](#).

7.66 Plugin module callback functions

7.66.1 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.

7.66.2 Function Documentation

7.66.2.1 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.

7.66.2.2 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.

7.66.2.3 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.

7.66.2.4 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.

7.66.2.5 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.

7.66.2.6 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.

7.66.2.7 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 afte 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.

7.66.2.8 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.

8 Orbiter API Class Documentation

8.1 ANIMATION Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for ANIMATION:

8.1.1 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](#)

8.2 ANIMATIONCOMP Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for ANIMATIONCOMP:

8.2.1 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](#)

8.3 ATMCONST Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for ATMCONST:

8.3.1 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

8.4 ATMOSPHERE Class Reference

```
#include <CelBodyAPI.h>
```

Collaboration diagram for ATMOSPHERE:

8.4.1 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.

8.4.2 Member Enumeration Documentation

8.4.2.1 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)

8.4.3 Constructor & Destructor Documentation

8.4.3.1 ATMOSPHERE::ATMOSPHERE (CELBODY2 * *body*)

Constructor. Creates an atmosphere instance for 'body'.

Parameters:

body pointer to celestial body

8.4.4 Member Function Documentation

8.4.4.1 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.

8.4.4.2 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 parameters 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.

8.4.4.3 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

8.5 ATMOSPHERE::PRM_IN Struct Reference

```
#include <CelBodyAPI.h>
```

8.5.1 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

8.6 ATMOSPHERE::PRM_OUT Struct Reference

```
#include <CelBodyAPI.h>
```

8.6.1 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/m³]

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[CeIBodyAPI.h](#)

8.7 ATMPARAM Struct Reference

```
#include <OrbiterAPI.h>
```

8.7.1 Detailed Description

Atmospheric parameters structure.

Public Attributes

- double **T**
temperature [K]
- double **p**
pressure [Pa]
- double **rho**
density [kg/m³]

The documentation for this struct was generated from the following file:

- Orbitersdk/include/[OrbiterAPI.h](#)

8.8 BEACONLIGHTSPEC Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for BEACONLIGHTSPEC:

8.8.1 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](#)

8.9 oapi::Brush Class Reference

```
#include <DrawAPI.h>
```

Inheritance diagram for oapi::Brush:

Collaboration diagram for oapi::Brush:

8.9.1 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.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 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](#)

8.10 CELBODY Class Reference

```
#include <CelBodyAPI.h>
```

Inheritance diagram for CELBODY:

8.10.1 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

8.10.2 Member Function Documentation

8.10.2.1 int CELBODY::Version () const [inline]

Return version number.

Returns:

Version number (1 for [CELBODY](#), 2 for [CELBODY2](#))

8.10.2.2 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*.

8.10.2.3 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](#).

8.10.2.4 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.

8.10.2.5 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.

8.10.2.6 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

8.11 CELBODY2 Class Reference

```
#include <CelBodyAPI.h>
```

Inheritance diagram for CELBODY2:



Collaboration diagram for CELBODY2:



8.11.1 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](#)

8.11.2 Constructor & Destructor Documentation**8.11.2.1 CELBODY2::CELBODY2 (OBJHANDLE *hCBody*)**

Constructor. Creates a [CELBODY2](#) instance for a celestial body.

Parameters:

hCBody body handle

8.11.2.2 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.

8.11.3 Member Function Documentation**8.11.3.1 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](#).

8.11.3.2 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.

8.11.3.3 OBJHANDLE CELBODY2::GetChild (DWORD *idx*) const

Returns for a child body in the solar system hierarchy.

Parameters:

idx child body index (≥ 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.

8.11.3.4 double CELBODY2::SidRotPeriod () const

Returns the sidereal period of the celestial body.

Returns:

Sidereal rotation period [s]

8.11.3.5 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.

8.11.3.6 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](#)

8.11.3.7 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.

8.11.3.8 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.

8.11.3.9 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);
```

8.11.3.10 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

8.12 COLOUR4 Struct Reference

```
#include <OrbiterAPI.h>
```

8.12.1 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

8.13 oapi::DrawingTool Class Reference

```
#include <DrawAPI.h>
```

Inheritance diagram for oapi::DrawingTool:

8.13.1 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](#)

8.14 ELEMENTS Struct Reference

```
#include <OrbiterAPI.h>
```

8.14.1 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](#)

8.15 ENGINESTATUS Struct Reference

```
#include <OrbiterAPI.h>
```

8.15.1 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](#)

8.16 EXHAUSTSPEC Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for EXHAUSTSPEC:

8.16.1 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

8.17 ExternMFD Class Reference

```
#include <MFDAPI.h>
```

8.17.1 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)

8.17.2 Constructor & Destructor Documentation

8.17.2.1 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.

8.17.2.2 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.

8.17.3 Member Function Documentation

8.17.3.1 `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) to create an identifier.

8.17.3.2 `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).

8.17.3.3 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.

8.17.3.4 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.

8.17.3.5 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\(\)](#).

8.17.3.6 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.

8.17.3.7 bool ExternMFD::ProcessButton (int *bt*, int *event*)

TODO

8.17.3.8 bool ExternMFD::SendKey (DWORD *key*)

TODO

8.17.3.9 bool ExternMFD::Resize (const MFDSPEC & *spec*)

TODO

8.17.3.10 bool ExternMFD::SetMode (int *mode*)

TODO

8.17.3.11 bool ExternMFD::OpenModeHelp () const

TODO

8.17.3.12 virtual void ExternMFD::clbkUpdate () [virtual]

TODO

8.17.3.13 virtual void ExternMFD::clbkRefreshDisplay (SURFHANDLE *hSurf*) [virtual]

TODO

8.17.3.14 virtual void ExternMFD::clbkRefreshButtons () [virtual]

TODO

8.17.3.15 virtual void ExternMFD::clbkFocusChanged (OBJHANDLE *hFocus*) [virtual]

TODO

The documentation for this class was generated from the following file:

- Orbitersdk/include/MFDAPIL.h

8.18 FogParam Struct Reference

```
#include <GraphicsAPI.h>
```

Collaboration diagram for FogParam:

8.18.1 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

8.19 oapi::Font Class Reference

```
#include <DrawAPI.h>
```

Inheritance diagram for oapi::Font:

Collaboration diagram for oapi::Font:

8.19.1 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.

8.19.2 Member Enumeration Documentation

8.19.2.1 enum oapi::Font::Style

Font decoration style.

See also:

`Font(int,bool,char*,Style)`

Enumerator:

NORMAL no decoration

BOLD boldface

ITALIC italic

UNDERLINE underlined

8.19.3 Constructor & Destructor Documentation

8.19.3.1 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.

8.19.4 Member Function Documentation

8.19.4.1 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](#)

8.20 oapi::GraphicsClient Class Reference

```
#include <GraphicsAPI.h>
```

Inheritance diagram for oapi::GraphicsClient:

Collaboration diagram for oapi::GraphicsClient:

8.20.1 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 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 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 tgy, SURFHANDLE src, DWORD flag=0) const`

Copy one surface into an area of another one.

- virtual bool `clbkBlt (SURFHANDLE tgt, DWORD tgtx, DWORD tgy, 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 tgy, 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 bool `clbkFillSurface (SURFHANDLE surf, DWORD col) const`

Fill a surface with a uniform colour.

- virtual bool `clbkFillSurface (SURFHANDLE surf, DWORD tgtx, DWORD tgy, 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.

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.

8.20.2 Constructor & Destructor Documentation

8.20.2.1 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](#))

8.20.2.2 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](#).

8.20.3 Member Function Documentation

8.20.3.1 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).

8.20.3.2 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.

8.20.3.3 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 Open-TextureFile.

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.

8.20.3.4 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.

8.20.3.5 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 ($>= 0$)

tex texture handle

Returns:

Should return *true* if operation successful, *false* otherwise.

Default action:

None, returns *false*.

8.20.3.6 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 ($>= 0$)

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").

8.20.3.7 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_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.

Default action:

None, returns *false*.

8.20.3.8 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](#)

8.20.3.9 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](#)

8.20.3.10 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](#)

8.20.3.11 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 (≥ 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.

8.20.3.12 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 (≥ 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.

8.20.3.13 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.

8.20.3.14 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](#)

8.20.3.15 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.

8.20.3.16 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* 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.

8.20.3.17 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.

8.20.3.18 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.

8.20.3.19 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.

8.20.3.20 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.

8.20.3.21 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

8.20.3.22 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.

8.20.3.23 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.

8.20.3.24 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.

8.20.3.25 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.

8.20.3.26 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);
```

8.20.3.27 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.

**8.20.3.28 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

8.20.3.29 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

**8.20.3.30 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

**8.20.3.31 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

8.20.3.32 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.

8.20.3.33 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.

8.20.3.34 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.

8.20.3.35 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](#)

8.20.3.36 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](#)

8.20.3.37 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](#)

8.20.3.38 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.

8.20.3.39 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](#)

8.20.3.40 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](#)

8.20.3.41 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.

8.20.3.42 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](#)

8.20.3.43 virtual bool oapi::GraphicsClient::clbkBlt (SURFHANDLE *tgt*, DWORD *tctx*, DWORD *tgy*, SURFHANDLE *src*, DWORD *flag* = 0) const [inline, virtual]

Copy one surface into an area of another one.

Parameters:

tgt target surface handle
tctx 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)`

8.20.3.44 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)

8.20.3.45 virtual bool oapi::GraphicsClient::clbkScaleBlt (SURFHANDLE *tgt*, DWORD *tgtx*, DWORD *tgyt*, DWORD *tgtw*, DWORD *tgh*, 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
tgh 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)

8.20.3.46 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\(SURFHANDLE,DWORD,DWORD,DWORD,DWORD,DWORD\)](#)

8.20.3.47 virtual bool oapi::GraphicsClient::clbkFillSurface (SURFHANDLE *surf*, DWORD *tgtx*, DWORD *tgyt*, DWORD *w*, DWORD *h*, DWORD *col*) const [inline, virtual]

Fill an area in a surface with a uniform colour.

Parameters:

surf surface handle

tgtx left edge of target rectangle

tgyt 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\(SURFHANDLE,DWORD\)](#)

8.20.3.48 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.

8.20.3.49 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](#)

8.20.3.50 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](#)

8.20.3.51 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](#)

See also:

[clbkReleaseFont](#), [oapi::Font](#)

8.20.3.52 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](#)

8.20.3.53 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](#)

8.20.3.54 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](#)

8.20.3.55 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](#)

8.20.3.56 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](#)

8.20.3.57 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

8.20.3.58 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

8.20.3.59 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.

8.20.3.60 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.

8.20.3.61 virtual void oapi::GraphicsClient::clbkPostCreation () [inline, protected, virtual]

Simulation startup finalisation.

Called at the beginning of a simulation session after the scenarion has been parsed and the logical object have been created.

Default action:

None

8.20.3.62 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](#)

8.20.3.63 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](#)

8.20.3.64 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.

8.20.3.65 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.)

8.20.3.66 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.

8.20.3.67 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`

8.20.3.68 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.

8.20.3.69 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.

8.20.3.70 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.

8.20.3.71 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.

8.20.3.72 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](#)

8.20.3.73 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

8.21 oapi::GraphicsClient::LABELLIST Struct Reference

```
#include <GraphicsAPI.h>
```

8.21.1 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

8.22 oapi::GraphicsClient::VIDEODATA Struct Reference

```
#include <GraphicsAPI.h>
```

8.22.1 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 **tryStencil**
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

8.23 GraphMFD Class Reference

```
#include <MFDAPI.h>
```

Inheritance diagram for GraphMFD:

Collaboration diagram for GraphMFD:

8.23.1 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**

8.23.2 Constructor & Destructor Documentation

8.23.2.1 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](#)

8.23.3 Member Function Documentation

8.23.3.1 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.

8.23.3.2 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.

8.23.3.3 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.

8.23.3.4 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.

8.23.3.5 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.

8.23.3.6 void GraphMFD::SetAxisTitle (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".

8.23.3.7 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.

8.23.3.8 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/MFDAPi.h

8.24 GROUPEDITSPEC Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for GROUPEDITSPEC:

8.24.1 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 (≥ 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](#)

8.25 HELPCONTEXT Struct Reference

```
#include <OrbiterAPI.h>
```

8.25.1 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](#)

8.26 HUDPARAM Union Reference

```
#include <OrbiterAPI.h>
```

8.26.1 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](#)

8.27 oapi::IVECTOR2 Union Reference

```
#include <DrawAPI.h>
```

8.27.1 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

8.28 LaunchpadItem Class Reference

```
#include <OrbiterAPI.h>
```

8.28.1 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

8.28.2 Member Function Documentation

8.28.2.1 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).

8.28.2.2 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).

8.28.2.3 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.

8.28.2.4 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.

8.28.2.5 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](#)

8.29 LightEmitter Class Reference

```
#include <OrbiterAPI.h>
```

Inheritance diagram for LightEmitter:

Collaboration diagram for LightEmitter:

8.29.1 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`

8.29.2 Constructor & Destructor Documentation

8.29.2.1 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`

8.29.2.2 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`

8.29.3 Member Function Documentation

8.29.3.1 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](#)

8.29.3.2 bool LightEmitter::IsActive () const

Returns activation status of light source.

Returns:

true if source is active, *false* otherwise.

See also:

[Activate](#)

8.29.3.3 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](#)

8.29.3.4 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:

[SetPosition](#), [SetPositionRef](#), [GetPositionRef](#)

8.29.3.5 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](#)

8.29.3.6 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](#)

8.29.3.7 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](#)

8.29.3.8 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](#)

8.29.3.9 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](#)

8.29.3.10 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](#)

8.29.3.11 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](#)

8.30 LISTENTRY Struct Reference

```
#include <OrbiterAPI.h>
```

8.30.1 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](#)

8.31 MATERIAL Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for MATERIAL:

8.31.1 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](#)

8.32 MATRIX3 Union Reference

```
#include <OrbiterAPI.h>
```

8.32.1 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](#)

8.33 MESHGROUP Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for MESHGROUP:

8.33.1 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](#)

8.34 MESHGROUP_TRANSFORM Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for MESHGROUP_TRANSFORM:

8.34.1 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

8.35 MESHGROUPEX Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for MESHGROUPEX:

8.35.1 Detailed Description

extended mesh group definition

Public Attributes

- **NTVERTEX * Vtx**
vertex list
- **WORD * Idx**
index list
- **DWORD nVtx**
vertex count
- **DWORD nIndex**
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

8.36 MFD Class Reference

```
#include <MFDAPI.h>
```

Inheritance diagram for MFD:

Collaboration diagram for MFD:

8.36.1 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**

8.36.2 Constructor & Destructor Documentation

8.36.2.1 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.

8.36.3 Member Function Documentation

8.36.3.1 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](#).

8.36.3.2 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.

8.36.3.3 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.

8.36.3.4 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.

8.36.3.5 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.

8.36.3.6 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.

8.36.3.7 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.

8.36.3.8 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.

8.36.3.9 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.

8.36.3.10 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*

8.36.3.11 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;
}

```

8.36.3.12 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.

8.36.3.13 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.

8.36.3.14 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.

8.36.3.15 **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/[MFD API.h](#)

8.37 MFD2 Class Reference

```
#include <MFD API.h>
```

Inheritance diagram for MFD2:

Collaboration diagram for MFD2:

8.37.1 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.

8.37.2 Constructor & Destructor Documentation

8.37.2.1 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.

8.37.3 Member Function Documentation

8.37.3.1 DWORD MFD2::GetWidth () const [inline]

Returns the [MFD](#) display width.

Returns:

[MFD](#) display width [pixel]

See also:

[GetHeight](#)

8.37.3.2 DWORD MFD2::GetHeight () const [inline]

Returns the [MFD](#) display height.

Returns:

[MFD](#) display height [pixel]

See also:

[GetWidth](#)

8.37.3.3 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](#).

8.37.3.4 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.

8.37.3.5 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.

8.37.3.6 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 MFD 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](#)

8.37.3.7 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.

8.37.3.8 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 MFD 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](#)

8.38 oapi::Module Class Reference

```
#include <ModuleAPI.h>
```

Inheritance diagram for oapi::Module:



```
graph TD; Object --> Module
```

Collaboration diagram for oapi::Module:



```
graph TD; Module --- Interface1
```

8.38.1 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.

8.38.2 Member Enumeration Documentation

8.38.2.1 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

8.38.3 Constructor & Destructor Documentation

8.38.3.1 oapi::Module::Module (HINSTANCE *hDLL*)

Creates a new [Module](#) instance.

Parameters:

hDLL DLL library instance handle (see [InitModule](#))

8.38.4 Member Function Documentation

8.38.4.1 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.

8.38.4.2 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.

8.38.4.3 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 afte 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](#)

8.38.4.4 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](#)

8.38.4.5 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.

8.38.4.6 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](#).

8.38.4.7 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.

8.38.4.8 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.

8.38.4.9 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.

8.38.4.10 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\(\)](#)

8.38.4.11 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

8.39 oapi::ModuleNV Class Reference

```
#include <ModuleAPI.h>
```

Inheritance diagram for oapi::ModuleNV:

8.39.1 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**

8.39.2 Constructor & Destructor Documentation

8.39.2.1 oapi::ModuleNV::ModuleNV (HINSTANCE *hDLL*)

Creates a new [ModuleNV](#) instance.

Parameters:

hDLL DLL library instance handle (see [InitModule](#))

8.39.3 Member Function Documentation

8.39.3.1 int oapi::ModuleNV::Version () const [inline]

[Module](#) interface version.

Returns:

version number

8.39.3.2 HINSTANCE oapi::ModuleNV::GetModule () const [inline]

Returns the module instance handle.

Returns:

[Module](#) instance handle.

8.39.3.3 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](#)

8.39.3.4 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](#)

8.39.3.5 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

8.40 NAVDATA Struct Reference

```
#include <OrbiterAPI.h>
```

8.40.1 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

8.41 NTVERTEX Struct Reference

```
#include <OrbiterAPI.h>
```

8.41.1 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](#)

8.42 ORBITPARAM Struct Reference

```
#include <OrbiterAPI.h>
```

8.42.1 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](#)

8.43 oapi::ParticleStream Class Reference

```
#include <GraphicsAPI.h>
```

Collaboration diagram for oapi::ParticleStream:

8.43.1 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**

8.43.2 Constructor & Destructor Documentation

8.43.2.1 oapi::ParticleStream::ParticleStream (**GraphicsClient** * *_gc*, **PARTICLESTREAMSPEC** * *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\(\)](#).

8.43.3 Member Function Documentation

8.43.3.1 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)

srlvl 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.

8.43.3.2 void oapi::ParticleStream::Attach (OBJHANDLE *hObj*, const VECTOR3 & *_pos*, const VECTOR3 & *_dir*, const double * *srlvl*)

Attach the stream to an object.

Parameters:

hObj object handle

_pos particle source point (object frame)

_dir particle direction (object frame)

srlvl pointer to particle generator level

Note:

This method uses fixed position and direction variables.

8.43.3.3 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.

8.43.3.4 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.

8.43.3.5 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.

8.43.3.6 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.

8.43.3.7 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.

8.43.3.8 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.

8.43.3.9 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

8.44 PARTICLESTREAMSPEC Struct Reference

```
#include <OrbiterAPI.h>
```

8.44.1 Detailed Description

Particle stream parameters.

Note:

The following mapping methods (LEVELMAP) between stream level L and opacity α 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)

8.44.2 Member Enumeration Documentation

8.44.2.1 enum PARTICLESTREAMSPEC::LTYPE

Particle lighting method.

Enumerator:

EMISSIVE emissive lighting (example: plasma stream)

DIFFUSE diffuse lighting (example: vapour stream)

8.44.2.2 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

8.44.3 Member Data Documentation

8.44.3.1 enum PARTICLESTREAMSPEC::LTYPE PARTICLESTREAMSPEC::ltype

Particle lighting method.

render lighting method

8.44.3.2 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](#)

8.45 oapi::Pen Class Reference

```
#include <DrawAPI.h>
```

Inheritance diagram for oapi::Pen:

Collaboration diagram for oapi::Pen:

8.45.1 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.

8.45.2 Constructor & Destructor Documentation

8.45.2.1 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

8.46 PointLight Class Reference

```
#include <OrbiterAPI.h>
```

Inheritance diagram for PointLight:

Collaboration diagram for PointLight:

8.46.1 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]

8.46.2 Constructor & Destructor Documentation

8.46.2.1 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

8.46.2.2 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

8.46.3 Member Function Documentation

8.46.3.1 double PointLight::GetRange () const [inline]

Returns the light source range.

Returns:

Light source range [m]

8.46.3.2 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.

8.46.3.3 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 :

$$\frac{I}{I_0} = \frac{1}{att_0 + datt_1 + d^2att_2}$$

8.46.3.4 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](#)

8.47 oapi::ScreenAnnotation Class Reference

```
#include <GraphicsAPI.h>
```

Collaboration diagram for oapi::ScreenAnnotation:

8.47.1 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.

8.47.2 Constructor & Destructor Documentation

8.47.2.1 oapi::ScreenAnnotation::ScreenAnnotation (GraphicsClient * *_gc*)

Constructs a new annotation object.

Parameters:

_gc pointer to graphics client

8.47.3 Member Function Documentation

8.47.3.1 virtual void oapi::ScreenAnnotation::SetText (char * *str*) [virtual]

Set the text to be displayed by the annotation object.

Parameters:

str character string

8.47.3.2 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.

8.47.3.3 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.

8.47.3.4 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

8.48 oapi::Sketchpad Class Reference

```
#include <DrawAPI.h>
```

8.48.1 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.

8.48.2 Member Enumeration Documentation

8.48.2.1 enum oapi::Sketchpad::TAlign_horizontal

Horizontal text alignment modes.

See also:

[SetTextAlign](#)

Enumerator:

- LEFT** align left
CENTER align center
RIGHT align right

8.48.2.2 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

8.48.2.3 enum oapi::Sketchpad::BkgMode

Background modes for text output.

See also:

[SetBackgroundMode](#)

Enumerator:

BK_TRANSPARENT transparent background

BK_OPAQUE opaque background

8.48.3 Constructor & Destructor Documentation

8.48.3.1 oapi::Sketchpad::Sketchpad (SURFHANDLE *s*)

Constructs a drawing object for a given surface.

Parameters:

s surface handle

8.48.4 Member Function Documentation

8.48.4.1 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](#)

8.48.4.2 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](#)

8.48.4.3 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](#)

8.48.4.4 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.

8.48.4.5 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.

8.48.4.6 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](#)

8.48.4.7 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](#)

8.48.4.8 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.

8.48.4.9 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](#)

8.48.4.10 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.

8.48.4.11 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.

8.48.4.12 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.

8.48.4.13 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)

8.48.4.14 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](#)

8.48.4.15 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](#)

8.48.4.16 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](#)

8.48.4.17 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](#)

8.48.4.18 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](#)

8.48.4.19 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](#)

8.48.4.20 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](#)

8.48.4.21 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](#)

8.48.4.22 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](#)

8.48.4.23 SURFHANDLE oapi::Sketchpad::GetSurface () const [inline]

Returns the surface associated with the drawing object.

Returns:

Surface handle

8.48.4.24 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

8.49 SpotLight Class Reference

```
#include <OrbiterAPI.h>
```

Inheritance diagram for SpotLight:

Collaboration diagram for SpotLight:

8.49.1 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`

8.49.2 Constructor & Destructor Documentation

8.49.2.1 `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.

8.49.2.2 `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.

8.49.3 Member Function Documentation

8.49.3.1 double SpotLight::GetUmbra () const [inline]

Returns the angular aperture of inner (maximum intensity) cone.

Returns:

Aperture of inner spotlight cone [rad]

See also:

[GetPenumbra](#)

8.49.3.2 double SpotLight::GetPenumbra () const [inline]

Returns the angular aperture of outer (zero intensity) cone.

Returns:

Aperture of outer spotlight cone [rad]

See also:

[GetUmbra](#)

8.49.3.3 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](#)

8.50 VECTOR3 Union Reference

```
#include <OrbiterAPI.h>
```

8.50.1 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

8.51 VESSEL Class Reference

```
#include <VesselAPI.h>
```

Inheritance diagram for VESSEL:

8.51.1 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 **GetAirspeed () const**
Returns magnitude of the freestream airflow velocity vector measured in ship-relative coordinates.
- 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 parameters 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 tgtN, 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

8.51.2 Constructor & Destructor Documentation

8.51.2.1 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](#)

8.51.3 Member Function Documentation

8.51.3.1 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](#)

8.51.3.2 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.

8.51.3.3 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*.

8.51.3.4 char* VESSEL::GetName () const

Returns the vessel's name.

Returns:

Pointer to vessel's name

See also:

[GetClassName](#)

8.51.3.5 char* VESSEL::GetClassName () const

Returns the vessel's class name.

Returns:

Pointer to vessel's class name.

See also:

[GetName](#)

8.51.3.6 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](#)

8.51.3.7 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](#)

8.51.3.8 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](#)

8.51.3.9 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](#)

8.51.3.10 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](#)**8.51.3.11 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](#)**8.51.3.12 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](#)

8.51.3.13 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](#)

8.51.3.14 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.

8.51.3.15 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](#)

8.51.3.16 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](#)

8.51.3.17 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](#)

8.51.3.18 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](#)

8.51.3.19 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](#)

8.51.3.20 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](#)

8.51.3.21 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_lng* = 0.1 and *mu_lat* = 0.5. If the vessel hasn't got wheels, *mu_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:

$$\mathbf{s}_{\text{lng}} = \mathbf{p}_0 - (\mathbf{p}_1 + \mathbf{p}_2)/2, \quad \mathbf{s}_{\text{lat}} = \mathbf{p}_2 - \mathbf{p}_1$$

See also:

[SetTouchdownPoints](#)

8.51.3.22 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 [m^2]

See also:

[SetCrossSections](#)

8.51.3.23 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 [m^2].

See also:

[GetCrossSections](#)

8.51.3.24 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 [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 shiedit utility in the SDK package allows to calculate the inertia tensor from a mesh, assuming a homogeneous mass distribution.

See also:

[SetPMI](#)

8.51.3.25 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 [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](#)

8.51.3.26 double VESSEL::GetGravityGradientDamping () const

Returns the vessel's damping coefficient for gravity field gradient-induced torque.

Returns:

Torque damping coefficient (≥ 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 ω_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](#)

8.51.3.27 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](#)

8.51.3.28 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](#)

8.51.3.29 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](#)

8.51.3.30 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`

8.51.3.31 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](#)

8.51.3.32 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)

8.51.3.33 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](#)

8.51.3.34 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](#)

8.51.3.35 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](#)

8.51.3.36 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](#)

8.51.3.37 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](#)

8.51.3.38 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](#)

8.51.3.39 void VESSEL::GetAngularAcc (VECTOR3 & *aacc*) const

Returns the vessel's current angular acceleration components around its principal axes.

Parameters:

→ *aacc* angular acceleration [rad/s²]

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](#)

8.51.3.40 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](#)

8.51.3.41 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](#)

8.51.3.42 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](#)

8.51.3.43 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](#)

8.51.3.44 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](#)

8.51.3.45 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](#)

8.51.3.46 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](#)

8.51.3.47 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](#)

8.51.3.48 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:

[SetADCrlMode](#), [CreateControlSurface](#), [CreateControlSurface2](#), [GetControlSurfaceLevel](#), [SetControlSurfaceLevel](#)

8.51.3.49 void VESSEL::SetADCrlMode (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](#)

8.51.3.50 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](#)

8.51.3.51 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](#)

8.51.3.52 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](#)

8.51.3.53 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](#)

8.51.3.54 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.

8.51.3.55 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](#)

8.51.3.56 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](#)

8.51.3.57 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](#)

8.51.3.58 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](#)

8.51.3.59 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](#)

8.51.3.60 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](#)

8.51.3.61 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](#)

8.51.3.62 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.

8.51.3.63 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](#)

8.51.3.64 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](#)

8.51.3.65 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](#)

8.51.3.66 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 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](#)

8.51.3.67 const 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](#)

8.51.3.68 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](#)

8.51.3.69 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](#)

8.51.3.70 double VESSEL::GetAtmDensity () const

Returns atmospheric density at current vessel position.

Returns:

Atmospheric density [kg/m³] 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](#)

8.51.3.71 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](#)

8.51.3.72 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 ρ and airflow velocity V . Dynamic pressure is an important aerodynamic parameter.

See also:

[GetAtmPressure](#), [GetAtmRef](#)

8.51.3.73 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](#)

8.51.3.74 double VESSEL::GetAirspeed () const

Returns magnitude of the freestream airflow velocity vector measured in ship-relative coordinates.

Returns:

Magnitude of airflow velocity vector [m/s]

Note:

This function also works in the absence of an atmosphere. At low altitudes, the returned value is a ground-speed equivalent. At high altitudes the value diverges from ground speed, since an atmospheric drag effect is assumed.

This function returns the length of the vector returned by [GetShipAirspeedVector\(\)](#).

See also:

[GetShipAirspeedVector](#), [GetMachNumber](#), [GetAtmRef](#)

8.51.3.75 bool VESSEL::GetHorizonAirspeedVector (VECTOR3 & v) const

Returns the airspeed vector in local horizon coordinates.

Parameters:

→ v airspeed vector on exit [m/s]

Returns:

false indicates error

Note:

This method returns the airspeed vector in the reference frame of the local horizon. x = longitudinal component, y = vertical component, z = latitudinal component.

See also:

[GetAirspeed](#), [GetShipAirspeedVector](#)

8.51.3.76 bool VESSEL::GetShipAirspeedVector (VECTOR3 & v) const

Returns the airspeed vector in vessel coordinates.

Parameters:

→ *v* airspeed vector on exit [m/s]

Returns:

false indicates error

Note:

This method returns the airspeed vector in local ship coordinates. x = lateral component, y = vertical component, z = longitudinal component.

See also:

[GetAirspeed](#), [GetHorizonAirspeedVector](#)

8.51.3.77 double VESSEL::GetAOA () const

Returns the current angle of attack.

Returns:

AOA (angle of attack) value [rad] in the range -Pi ... +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](#)

8.51.3.78 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 ... +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](#)

8.51.3.79 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_x* , *v_y* , *v_z*) be the unit vector of freestream air flow in vessel coordinates. Then the reference area is calculated as *S* = *v_z* *C_z* + *v_y* *C_y* for a LIFT_VERTICAL airfoil, and as *S* = *v_z* *C_z* + *v_x* *C_x* for a LIFT_HORIZONTAL airfoil, where *C_x* , *C_y* , *C_z* are the vessel cross-sections in x, y and z direction, respectively.

The wing aspect ratio is defined as *A* = *b*² /*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](#)

8.51.3.80 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 [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](#)

8.51.3.81 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 [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](#)

8.51.3.82 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](#)

8.51.3.83 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](#)

8.51.3.84 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](#)

8.51.3.85 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](#)

8.51.3.86 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 [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](#)

8.51.3.87 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 [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](#)

8.51.3.88 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](#)

8.51.3.89 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](#)

8.51.3.90 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](#)

8.51.3.91 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](#)

8.51.3.92 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](#)

8.51.3.93 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](#)

8.51.3.94 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](#)

8.51.3.95 void VESSEL::ClearVariableDragElements () const

Removes all drag elements defined with CreateVariableDragElement.

See also:

[CreateVariableDragElement](#)

8.51.3.96 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](#)

8.51.3.97 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](#)

8.51.3.98 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](#)

8.51.3.99 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](#)

8.51.3.100 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 ~ 3.1 for elliptic wings, ~ 2.8 for tapered wings, and ~ 2.5 for rectangular wings. Default is 2.8.

See also:

[SetWingEffectiveness](#), [GetWingAspect](#), [CreateAirfoil](#)

8.51.3.101 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: ~ 3.1 for elliptic wings, ~ 2.8 for tapered wings, ~ 2.5 for rectangular wings.

See also:

[GetWingEffectiveness](#), [SetWingAspect](#), [CreateAirfoil](#)

8.51.3.102 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](#)

8.51.3.103 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](#)

8.51.3.104 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](#)

8.51.3.105 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 no airfoils have been defined.

The default value is 0.

See also:

[GetPitchMomentScale](#), [GetYawMomentScale](#), [CreateAirfoil](#)

8.51.3.106 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](#)

8.51.3.107 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 no airfoils have been defined.

The default value is 0.

See also:

[SetPitchMomentScale](#), [GetYawMomentScale](#), [CreateAirfoil](#)

8.51.3.108 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](#)

8.51.3.109 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](#)

8.51.3.110 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](#)

8.51.3.111 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](#)

8.51.3.112 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](#)

8.51.3.113 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](#)

8.51.3.114 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](#)

8.51.3.115 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](#)

8.51.3.116 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](#)

8.51.3.117 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](#)

8.51.3.118 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](#)**8.51.3.119 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](#)**8.51.3.120 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](#)

8.51.3.121 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](#)

8.51.3.122 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](#)

8.51.3.123 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](#)

8.51.3.124 PROPELLANT_HANDLE VESSEL::GetPropellantHandleByIndex (DWORD *idx*) const

Returns the handle of a propellant resource for a given index.

Parameters:

idx propellant resource index (≥ 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](#)

8.51.3.125 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](#)

8.51.3.126 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 (≥ 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](#)

8.51.3.127 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](#)

8.51.3.128 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 (≥ 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](#)

8.51.3.129 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](#)

8.51.3.130 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](#)

8.51.3.131 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](#)

8.51.3.132 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](#)

8.51.3.133 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](#)

8.51.3.134 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](#)

8.51.3.135 PROPELLANT_HANDLE VESSEL::GetDefaultPropellantResource () const

Returns the handle for the vessel's default propellant resource.

Returns:

Default propellant resource handle

See also:

[SetDefaultPropellantResource](#)

8.51.3.136 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](#)

8.51.3.137 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](#)

8.51.3.138 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](#)

8.51.3.139 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](#)

8.51.3.140 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](#)

8.51.3.141 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](#)

8.51.3.142 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](#)

8.51.3.143 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](#)

8.51.3.144 DWORD VESSEL::GetThrusterCount () const

Returns the number of thrusters currently defined.

Returns:

Number of logical thruster definitions.

See also:

[CreateThruster](#), [GetThrusterHandleByIndex](#)

8.51.3.145 THRUSTER_HANDLE VESSEL::GetThrusterHandleByIndex (DWORD idx) const

Returns the handle of a thruster specified by its index.

Parameters:

idx thruster index (≥ 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](#)

8.51.3.146 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](#)

8.51.3.147 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](#)

8.51.3.148 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](#)

8.51.3.149 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](#)

8.51.3.150 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](#)

8.51.3.151 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](#)

8.51.3.152 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](#)

8.51.3.153 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](#)

8.51.3.154 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](#)

8.51.3.155 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 a 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](#)

8.51.3.156 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](#)

8.51.3.157 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](#)

8.51.3.158 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](#)

8.51.3.159 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](#)

8.51.3.160 void VESSEL::SetThrusterIsp (THRUSTER_HANDLE *th*, double *isp0*, double *isp_ref*, double *p_ref* = 101.4×10^3) 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](#)

8.51.3.161 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](#)

8.51.3.162 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](#)

8.51.3.163 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](#)

8.51.3.164 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\(\)](#)

8.51.3.165 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\(\)](#)

8.51.3.166 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.

8.51.3.167 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](#)

8.51.3.168 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.

8.51.3.169 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

8.51.3.170 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

8.51.3.171 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](#)

8.51.3.172 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](#)

8.51.3.173 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 (≥ 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](#)

8.51.3.174 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](#)

8.51.3.175 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](#)

8.51.3.176 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

8.51.3.177 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](#).

8.51.3.178 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.)

8.51.3.179 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](#)

8.51.3.180 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](#)

8.51.3.181 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](#)

8.51.3.182 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](#)

8.51.3.183 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](#)

8.51.3.184 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](#)

8.51.3.185 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_-](#)

8.51.3.186 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](#)

8.51.3.187 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](#)

8.51.3.188 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.

8.51.3.189 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](#)

8.51.3.190 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](#)

8.51.3.191 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](#)

8.51.3.192 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](#)

8.51.3.193 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](#)

8.51.3.194 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](#)

8.51.3.195 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](#)

8.51.3.196 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](#)

8.51.3.197 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](#)

8.51.3.198 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](#)

8.51.3.199 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](#)

8.51.3.200 DWORD VESSEL::GetNavCount () const

Returns the number of NAV radio receivers.

Returns:

Number of NAV receivers (≥ 0)

See also:

[InitNavRadios](#)

8.51.3.201 bool VESSEL::SetNavChannel (DWORD *n*, DWORD *ch*) const

Sets the channel of a NAV radio receiver.

Parameters:

n receiver index (≥ 0)

ch channel (≥ 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](#)

8.51.3.202 DWORD VESSEL::GetNavChannel (DWORD *n*) const

Returns the current channel setting of a NAV radio receiver.

Parameters:

n receiver index (≥ 0)

Returns:

Receiver channel [0..639]. If index *n* is out of range, the return value is 0.

See also:

[GetNavRecvFreq](#), [SetNavChannel](#)

8.51.3.203 float VESSEL::GetNavRecvFreq (DWORD *n*) const

Returns the current radio frequency of a NAV radio receiver.

Parameters:

n receiver index (≥ 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](#)

8.51.3.204 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](#)

8.51.3.205 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](#)

8.51.3.206 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](#)

8.51.3.207 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](#)

8.51.3.208 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](#)

8.51.3.209 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](#)

8.51.3.210 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 (≥ 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.

8.51.3.211 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](#)

8.51.3.212 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](#)

8.51.3.213 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](#)

8.51.3.214 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](#)

8.51.3.215 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](#)

8.51.3.216 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\(\)](#).

8.51.3.217 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 ≥ 0 . The left and right ranges should be $< \pi$. The up and down ranges should be $< \pi/2$.

The default values are 0.8π for left and right ranges, and 0.4π for up and down ranges.

See also:

[SetCameraShiftRange](#), [SetCameraMovement](#)

8.51.3.218 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](#)

8.51.3.219 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](#)

8.51.3.220 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 [ClearMeshes\(\)const](#) will be removed, and *retain_anim* will have a default value of false.

8.51.3.221 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](#)

8.51.3.222 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](#)

8.51.3.223 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 (≥ 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](#)

8.51.3.224 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 (≥ 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](#)

8.51.3.225 bool VESSEL::DelMesh (UINT *idx*, bool *retain_anim* = false) const

Remove a mesh from the vessel's mesh list.

Parameters:

idx mesh list index (≥ 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](#)

8.51.3.226 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 (≥ 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](#)

8.51.3.227 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](#)

8.51.3.228 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](#)

8.51.3.229 UINT VESSEL::GetMeshCount () const

Number of meshes.

Returns the number of meshes currently defined for the vessel

Returns:

mesh count (≥ 0)

8.51.3.230 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](#)

8.51.3.231 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](#)

8.51.3.232 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.

8.51.3.233 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.

8.51.3.234 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.

8.51.3.235 WORD VESSEL::GetMeshVisibilityMode (UINT *idx*) const

Returns the visibility flags for a vessel mesh.

Parameters:

idx mesh index (≥ 0)

Returns:

Visibility mode flags (see [SetMeshVisibilityMode](#) for possible values).

See also:

[SetMeshVisibilityMode](#), Vessel mesh visibility flags

8.51.3.236 void VESSEL::SetMeshVisibilityMode (UINT *idx*, WORD *mode*) const

Set the visibility flags for a vessel mesh.

Parameters:

idx mesh index (≥ 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](#)

8.51.3.237 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*.

8.51.3.238 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 (≥ 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](#)

8.51.3.239 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](#)

8.51.3.240 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](#)

8.51.3.241 `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](#)

8.51.3.242 `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](#)

8.51.3.243 `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](#)

8.51.3.244 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](#)

8.51.3.245 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.

8.51.3.246 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.

8.51.3.247 bool VESSEL::Recording () const

Flag for active recording session.

Returns:

true if flight recording is active, *false* otherwise.

See also:

[Playback](#), [RecordEvent](#)

8.51.3.248 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](#)

8.51.3.249 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](#)

8.51.3.250 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](#)

8.51.3.251 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](#)

8.51.3.252 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).

8.51.3.253 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_{vessel}* is the vessel's global position.

This transformation can be directly performed by a call to Local2Global.

See also:

[Local2Global](#), [SetRotationMatrix](#), [GlobalRot](#)

8.51.3.254 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](#)

8.51.3.255 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](#)

8.51.3.256 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](#)

8.51.3.257 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](#)

8.51.3.258 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](#)

8.51.3.259 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](#)

8.51.3.260 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}_{ref} is the reference body's global position.

See also:

[GetRotationMatrix](#), [Global2Local](#), [Local2Global](#), [GetGravityRef](#)

8.51.3.261 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](#)

8.51.3.262 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](#)

8.51.3.263 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](#)

8.51.3.264 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](#)

8.51.3.265 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](#)

8.51.3.266 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](#)

8.51.3.267 UINT VESSEL::DockCount () const

Returns the number of docking ports defined for the vessel.

Returns:

Number of docking ports.

See also:

[CreateDock](#), [DelDock](#), [ClearDockDefinitions](#)

8.51.3.268 DOCKHANDLE VESSEL::GetDockHandle (UINT *n*) const

Returns a handle to a docking port.

Parameters:

- n* docking port index (≥ 0)

Returns:

Dock handle, or NULL if index is out of range.

See also:

[CreateDock](#), [DelDock](#), [SetDockParams](#), [GetDockParams](#), [GetDockStatus](#), [oapiGetDockHandle](#)

8.51.3.269 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](#)

8.51.3.270 UINT VESSEL::DockingStatus (UINT *port*) const

Returns a status flag for a docking port.

Parameters:

port docking port index (≥ 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](#)

8.51.3.271 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 (≥ 0)

tgn docking port index on target (≥ 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](#)

8.51.3.272 bool VESSEL::Undock (UINT *n*, const OBJHANDLE *exclude* = 0) const

Release a docked vessel from a docking port.

Parameters:

n docking port index (≥ 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](#)

8.51.3.273 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.

8.51.3.274 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](#)

8.51.3.275 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](#)

8.51.3.276 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.

8.51.3.277 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](#)

8.51.3.278 void VESSEL::GetAttachmentParams (ATTACHMENTHANDLE *attachment*, VECTOR3 & *pos*, VECTOR3 & *dir*, VECTOR3 & *rot*) const

Retrieve the parameters of an attachment point.

Parameters:

← *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](#)

8.51.3.279 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](#)

8.51.3.280 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](#)

8.51.3.281 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](#)

8.51.3.282 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 (≥ 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](#)

8.51.3.283 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 (≥ 0)

Returns:

Attachment handle, or NULL if index out of range.

See also:

[CreateAttachment](#), [SetAttachmentParams](#), [GetAttachmentParams](#), [GetAttachmentId](#), [GetAttachmentStatus](#), [AttachmentCount](#), [GetAttachmentIndex](#), [AttachChild](#), [DetachChild](#)

8.51.3.284 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](#)

8.51.3.285 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](#)

8.51.3.286 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](#)

8.51.3.287 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](#)

8.51.3.288 **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\(THRUSTER_HANDLE,double,double,SURFHANDLE\)const](#),
[DelExhaust](#),
[CreateThruster](#),
[SetThrusterRef](#),
[SetThrusterDir](#),
[SetThrusterLevel](#),
[oapiRegisterExhaustTexture](#)

8.51.3.289 **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.

8.51.3.290 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](#)

8.51.3.291 DWORD VESSEL::GetExhaustCount () const

Returns the number of exhaust render definitions for the vessel.

Returns:

Number of exhaust render definitions

See also:

[AddExhaust](#), [DelExhaust](#)

8.51.3.292 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](#)

8.51.3.293 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*.

8.51.3.294 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](#)

8.51.3.295 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](#)

8.51.3.296 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](#)

**8.51.3.297 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](#)

**8.51.3.298 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](#)

**8.51.3.299 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](#)

8.51.3.300 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](#)

8.51.3.301 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](#)

8.51.3.302 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](#)

8.51.3.303 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](#)

8.51.3.304 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](#)

8.51.3.305 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](#)

8.51.3.306 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](#)

8.51.3.307 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](#)

8.51.3.308 void VESSEL::ClearBeacons ()

Remove all beacon definitions from the vessel.

See also:

[AddBeacon](#), [DelBeacon](#)

8.51.3.309 const BEACONLIGHTSPEC* VESSEL::GetBeacon (DWORD *idx*) const

Returns a pointer to one of the vessel's beacon specifications.

Parameters:

idx beacon list index (≥ 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.

8.51.3.310 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

$$I = \frac{1}{att_0 + datt_1 + d^2att_2}$$

8.51.3.311 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

$$I = \frac{1}{att_0 + datt_1 + d^2att_2}$$

8.51.3.312 DWORD VESSEL::LightEmitterCount () const

Returns the number of light sources defined for the vessel.

Returns:

Number of light sources.

8.51.3.313 const LightEmitter* VESSEL::GetLightEmitter (DWORD i) const

Returns a pointer to a light source object identified by index.

Parameters:

i emitter index (≥ 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](#)

8.51.3.314 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](#)

8.51.3.315 void VESSEL::ClearLightEmitters () const

Remove all light sources defined for the vessel.

See also:

[AddPointLight](#), [AddSpotLight](#), [LightEmitterCount](#)

8.51.3.316 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](#)

8.51.3.317 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](#)

8.51.3.318 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](#)

8.51.3.319 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

8.51.3.320 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](#)

8.51.3.321 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](#)

8.51.3.322 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](#)

8.51.3.323 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 (≥ 0)

ch channel (≥ 0)

Returns:

false on error (index out of range), *true* otherwise

8.51.3.324 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 (≥ 0)

Returns:

Receiver channel [0..639]. If index *n* is out of range, the return value is 0.

8.51.3.325 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](#).

8.51.3.326 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](#)

8.51.3.327 void VESSEL::SetMeshVisibleInternal (UINT *idx*, bool *visible*) const

Marks a mesh as visible from internal cockpit view.

Parameters:

idx mesh index (≥ 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](#)

8.51.3.328 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](#)

8.51.3.329 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](#)

8.51.3.330 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](#)

8.52 VESSEL2 Class Reference

```
#include <VesselAPI.h>
```

Inheritance diagram for VESSEL2:

Collaboration diagram for VESSEL2:

8.52.1 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.

8.52.2 Constructor & Destructor Documentation

8.52.2.1 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`

8.52.3 Member Function Documentation

8.52.3.1 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.

8.52.3.2 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.

8.52.3.3 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](#).

8.52.3.4 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](#).

8.52.3.5 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.

8.52.3.6 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.

8.52.3.7 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](#).

8.52.3.8 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](#)

8.52.3.9 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.

8.52.3.10 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.

8.52.3.11 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.

8.52.3.12 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](#)

8.52.3.13 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.

8.52.3.14 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.

8.52.3.15 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.

8.52.3.16 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.

8.52.3.17 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.

8.52.3.18 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 (≥ 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.

8.52.3.19 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](#)

8.52.3.20 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.

8.52.3.21 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.

8.52.3.22 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](#)

8.52.3.23 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 (≥ 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](#)

8.52.3.24 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](#)).

8.52.3.25 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](#)

8.52.3.26 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 (≥ 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](#)

8.52.3.27 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](#)

8.52.3.28 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

8.53 VESSEL3 Class Reference

```
#include <VesselAPI.h>
```

Inheritance diagram for VESSEL3:

Collaboration diagram for VESSEL3:

8.53.1 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` hDefaultTex)
HUD render notification.
- virtual void `clbkGetRadiationForce` (const `VECTOR3` &mflux, `VECTOR3` &F, `VECTOR3` &pos)
Returns force due to radiation pressure.

8.53.2 Constructor & Destructor Documentation

8.53.2.1 VESSEL3::VESSEL3 (`OBJHANDLE` *hVessel*, int *fmodel* = 1)

Creates a `VESSEL3` interface for a vessel object.

See also:

[VESSEL2](#)

8.53.3 Member Function Documentation

8.53.3.1 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.

8.53.3.2 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

8.53.3.3 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 (≥ 0)

nmesh panel mesh index (≥ 0)

ngroup mesh group index (≥ 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](#).

8.53.3.4 int VESSEL3::RegisterPanelArea (PANELHANDLE *hPanel*, int *id*, const RECT & *pos*, const RECT & *texpos*, int *draw_event*, int *mouse_event*, int *bemode*)

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)

texpos area boundary (texture coordinates)

draw_event event flags for redraw event triggers

mouse_event event flags for mouse event triggers

bemode flag for texture background provided to redraw callback function

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 update one of the panel textures, the texture handle should be passed in *hTgt*, and the affected area (in pixels) of the texture bitmap should be passed in *texpos*.

8.53.3.5 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
mouse_event event flags for mouse event triggers
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.

8.53.3.6 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](#)

8.53.3.7 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 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:

[RegisterPanelArea](#), [oapiGetDC](#), [oapiReleaseDC](#), [oapiTriggerPanelRedrawArea](#)

8.53.3.8 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.

8.53.3.9 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 (≥ 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.

8.53.3.10 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.

8.53.3.11 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.

8.53.3.12 virtual void VESSEL3::clbkGetRadiationForce (const VECTOR3 & *mflux*, VECTOR3 & *F*, VECTOR3 & *pos*) [virtual]

Returns force due to radiation pressure.

Parameters:

← *mflux* momentum flux vector [N/m²] at current spacecraft position, transformed into vessel frame
→ *F* radiation force vector [N] in vessel frame
→ *pos* force attack point [m] in vessel frame

Default action:

Sets *F* = *mflux* * size² * *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](#)

8.54 VESSELSTATUS Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for VESSELSTATUS:

8.54.1 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

8.54.2 Member Data Documentation

8.54.2.1 int VESSELSTATUS::status

flight status indicator

Note:

- 0=active (freeflight)
- 1=inactive (landed)

8.54.2.2 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

8.54.2.3 DWORD VESSELSTATUS::flag[10]

additional integer and bitflag parameters

flag[0]&1:

- 0: ignore 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](#)

8.55 VESSELSTATUS2 Struct Reference

```
#include <OrbiterAPI.h>
```

Collaboration diagram for VESSELSTATUS2:

8.55.1 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

8.55.2 Member Data Documentation

8.55.2.1 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](#)

8.55.2.2 int VESSELSTATUS2::status

vessel status indicator

Note:

- 0=active (freeflight)
- 1=inactive (landed)

8.55.2.3 VECTOR3 VESSELSTATUS2::arot

vessel orientation against ecliptic frame

arot (α, β, γ) 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.

8.55.2.4 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)

8.55.2.5 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)

8.55.2.6 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](#)

8.56 VESSELSTATUS2::DOCKINFOSPEC Struct Reference

```
#include <OrbiterAPI.h>
```

8.56.1 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](#)

8.57 VESSELSTATUS2::FUELSPEC Struct Reference

```
#include <OrbiterAPI.h>
```

8.57.1 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](#)

8.58 VESSELSTATUS2::THRUSTSPEC Struct Reference

```
#include <OrbiterAPI.h>
```

8.58.1 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](#)

9 Orbiter API File Documentation

9.1 Orbitersdk/include/CelBodyAPI.h File Reference

9.1.1 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

9.2 Orbitersdk/include/DrawAPI.h File Reference

9.2.1 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.

9.3 Orbitersdk/include/MFDAPi.h File Reference

9.3.1 Detailed Description

Class interfaces for [MFD](#) instruments and [MFD](#) modes.

```
#include "OrbiterAPI.h"
```

Include dependency graph for MFDAPi.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.

9.4 Orbitersdk/include/OrbiterAPI.h File Reference

9.4.1 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 **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_DOCKINFORLIST** 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` (`I`=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

- `#define NAVMODE_HLEVEL 2`
"Kill rotation" mode
- `#define NAVMODE_PROGRADE 3`
"Hold level with horizon" mode
- `#define NAVMODE_RETROGRADE 4`
"Prograde" mode
- `#define NAVMODE_NORMAL 5`
"Retrograde" mode
- `#define NAVMODE_ANTINORMAL 6`
"Normal to orbital plane" mode
- `#define NAVMODE_HOLDALT 7`
"Anti-normal to orbital plane" 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** 0x00
Don't generate redraw events.
- #define **PANEL_REDRAW_ALWAYS** 0x01
Generate event at each frame.
- #define **PANEL_REDRAW_MOUSE** 0x02
Generate event on mouse event.
- #define **PANEL_REDRAW_INIT** 0x03
Initialisation event.
- #define **PANEL_REDRAW_USER** 0x04
User-generated event.
- #define **PANEL_MOUSE_IGNORE** 0x00
Don't generate mouse events.
- #define **PANEL_MOUSE_LBDOWN** 0x01
Left button down event.
- #define **PANEL_MOUSE_RBDOWN** 0x02
Right button down event.
- #define **PANEL_MOUSE_LBUP** 0x04
Left button release event.
- #define **PANEL_MOUSE_RBUP** 0x08

Right button release event.

- #define **PANEL_MOUSE_LBPRESSED** 0x10

Left button down (continuous).

- #define **PANEL_MOUSE_RBPRESSED** 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

- #define **PANEL_MAP_BACKGROUND** 0x01

- #define **PANEL_MAP_CURRENT** 0x02

- #define **PANEL_MAP_BGONREQUEST** 0x03

- #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_SURFACELEVEL** 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

- #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_NUMPADERTER** 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
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_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

- #define **OAPI_LKEY_LPRCSBack** 59
linear RCS: accelerate forward 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
toggle on-screen info block on/off
- #define **OAPI_LKEY_ToggleFPS** 94
toggle frame rate display on/off
- #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**
- typedef void * **VISHANDLE**
- typedef void * **MESHHANDLE**
- typedef int * **DEVMESHHANDLE**

- `typedef void * SURFHANDLE`
- `typedef void * PANELHANDLE`
- `typedef void * FILEHANDLE`
- `typedef void * INTERPRETERHANDLE`
- `typedef void * THRUSTER_HANDLE`
- `typedef void * THGROUP_HANDLE`
- `typedef void * PROPELLANT_HANDLE`
- `typedef void * PSTREAM_HANDLE`
- `typedef void * DOCKHANDLE`
- `typedef void * ATTACHMENTHANDLE`
- `typedef void * AIRFOILHANDLE`
- `typedef void * CTRLSURFHANDLE`
- `typedef void * NAVHANDLE`
- `typedef void * ANIMATIONCOMPONENT_HANDLE`
- `typedef void * LAUNCHPADITEM_HANDLE`
- `typedef void * NOTEHANDLE`
- `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, MESHS, MODULES }`
- `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

- 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 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 [oapiGetAirspeed](#) (**OBJHANDLE** hVessel, double *airspeed)
Returns a vessel's airspeed w.r.t. the closest planet or moon.
- 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 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 `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.
- OAPIFUNC INTERPRETERHANDLE `oapiCreateInterpreter ()`
Returns a handle to a new interpreter instance.
- OAPIFUNC int `oapiDelInterpreter (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](#) * [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.
- 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 [oapiBroadcastMFDMassage](#) (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 bemode=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 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 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 void [oapiColourFill](#) ([SURFHANDLE](#) tgt, DWORD fillcolor, int tgtx=0, int tgty=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 [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(*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)
- 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 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 DWORD [oapiGetStationCount](#) ()
- 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.
- [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

9.5 Orbitersdk/include/VesselAPI.h File Reference

9.5.1 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.*

Defines

- #define FRAME_ECL 0
- #define FRAME_EQU 1

10 Orbiter API Example Documentation

10.1 clbkLoadStateEx.cpp

Example of an overloaded `VESSEL2::clbkLoadStateEx` method.

```

1 class MyVessel: public VESSEL2 {
2 public:
3     ...
4     clbkLoadStateEx (FILEHANDLE scn, void *status);
5     ...
6 };
7
8 void MyVessel::clbkLoadStateEx (FILEHANDLE scn, void *status)
9 {
10    char *line;
11    int my_value;
12
13    while (oapiReadScenario_nextline (scn, line)) {
14        if (!strnicmp (line, "my_option", 9)) { // custom item
15            sscanf (line+9, "%d", &my_value);
16        } else if (...) { // more items
17            ...
18        } else { // anything not recognised is passed on to Orbiter
19            ParseScenarioLineEx (line, vs);
20        }
21    }
22 }
```

10.2 clbkPreStep.cpp

Example of an overloaded `VESSEL2::clbkPreStep` method.

```

1 class MyVessel: public VESSEL2 {
2 public:
3     ...
4     clbkPreStep (double simt, double simdt, double mjd);
5     ...
6 };
7
8 void MyVessel::clbkPreStep (double simt, double simdt, double mjd)
9 {
10    double F = mass * dv/simdt;
11    AddForce(_V(0,0,F), _V(0,0,0));
12 }
```

10.3 clbkSetStateEx.cpp

Example of an overloaded `VESSEL2::clbkSetStateEx` method.

```

1 class MyVessel: public VESSEL2 {
2 public:
```

```

3     ...
4     clbkSetStateEx (const void *status);
5     ...
6 };
7
8 void MyVessel::clbkSetStateEx (const void *status)
9 {
10    // specialised vessel initialisations
11    // ...
12
13    // default initialisation:
14    DefSetStateEx (status);
15 }

```

11 VESSEL2.cpp

Example for constructing and destroying an overloaded **VESSEL2** (p. 482) 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 }

```

12 Orbiter API Page Documentation

12.1 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.

12.1.1 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 [InitInstance\(\)](#) 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.

12.1.2 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](#)

12.2 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](#)

12.3 Deleted and obsolete functions and methods

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

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

12.4 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.

12.4.1 Contents

[Elliptic orbits](#)

[The orbit in space](#)

[Kepler's equation](#)

12.5 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:

$$\mu = GM$$

12.5.1 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 (ν) 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

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

12.6 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 Υ and node N_1 is the *longitude of the ascending node* (θ).

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

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.

12.6.1 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

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

12.7 Kepler's equation

12.7.1 True and eccentric anomaly

To find the position P of an orbiting body at some time t , we need to find its true anomaly ν 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 ν 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*.

12.7.2 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.

12.8 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.

12.8.1 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::ParticleSystem](#)
- Overload the particle stream-related callback methods of [oapi::GraphicsClient](#), including
 - [oapi::GraphicsClient::clbkCreateParticleSystem\(\)](#)
 - [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.

12.8.2 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.

12.8.3 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.

12.9 Vessel module concepts

12.9.1 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.

12.9.2 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 Orbitersdk\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).

12.10 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

12.11 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::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.

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 VEC-  
TOR3&,SURFHANDLE)const
```

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 [VES-
SEL::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 `oapiGetAtmPressureDensity` This function has been replaced by `oapiGetAtm`.

Member `oapiGetFocusAtmPressureDensity` This function has been replaced by `oapiGetAtm`.

Member `oapiGetMFDModeSpec` This function has been replaced by `oapiGetMFDModeSpecEx`

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 oapiRegisterMFDMode This function has been replaced by [oapiRegisterMFDMode\(MFDMODESPECEX&\)](#).

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](#).

12.12 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 *trn)

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.

Index

~CELBODY2
 CELBODY2, 206

~ExternMFD
 ExternMFD, 215

~GraphicsClient
 oapi::GraphicsClient, 227

_M
 vec, 22

_V
 vec, 22

Activate
 LightEmitter, 270

ActivateNavmode
 VESSEL, 374

Active
 ExternMFD, 215

AddAnimationComponent
 VESSEL, 448

AddBeacon
 VESSEL, 473

AddExhaust
 VESSEL, 465–467

AddExhaustStream
 VESSEL, 470, 471

AddForce
 VESSEL, 402

AddGraph
 GraphMFD, 259

AddMesh
 VESSEL, 439, 440

AddParticleStream
 VESSEL, 470

AddPlot
 GraphMFD, 259

AddPointLight
 VESSEL, 475

AddReentryStream
 VESSEL, 471

AddSpotLight
 VESSEL, 475

Aerodynamic control surface types, 34

AIRCTRL_AILERON
 airctrltype, 34

AIRCTRL_AXIS_AUTO
 airctrlaxis, 35

AIRCTRL_ELEVATOR
 airctrltype, 34

AIRCTRL_ELEVATORTRIM
 airctrltype, 34

AIRCTRL_FLAP

 airctrltype, 34

AIRCTRL_RUDDER
 airctrltype, 34

AIRCTRL_RUDDERTRIM
 airctrltype, 34

AIRCTRL_TYPE
 airctrltype, 34

airctrlaxis
 AIRCTRL_AXIS_AUTO, 35

airctrltype
 AIRCTRL_AILERON, 34
 AIRCTRL_ELEVATOR, 34
 AIRCTRL_ELEVATORTRIM, 34
 AIRCTRL_FLAP, 34
 AIRCTRL_RUDDER, 34
 AIRCTRL_RUDDERTRIM, 34
 AIRCTRL_TYPE, 34

Airfoil orientation, 34

AIRFOIL_ORIENTATION
 airfoilliftdir, 34

AIRFOILHANDLE
 handle, 18

airfoilliftdir
 AIRFOIL_ORIENTATION, 34
 LIFT_HORIZONTAL, 34
 LIFT_VERTICAL, 34

ANIMATION, 190

Animation flags, 32

ANIMATIONCOMP, 191

ANIMATIONCOMPONENT_HANDLE
 handle, 18

Annotations
 oapiAnnotationSetColour, 166
 oapiAnnotationSetPos, 166
 oapiAnnotationSetSize, 167
 oapiAnnotationSetText, 167
 oapiCreateAnnotation, 167
 oapiDelAnnotation, 168

arot
 VESSELSTATUS2, 514

ATMCONST, 192

ATMOSPHERE, 193

 ATMOSPHERE, 194
 clbkConstants, 195
 clbkName, 194
 clbkParams, 195
 PRM_ALT, 194
 PRM_AP, 194
 PRM_F, 194
 PRM_FBR, 194

PRM_IN_FLAG, 194
PRM_LAT, 194
PRM_LNG, 194
ATMOSPHERE::PRM_IN, 196
ATMOSPHERE::PRM_OUT, 196
ATMPARAM, 197
Attach
 oapi::ParticleStream, 304, 305
AttachChild
 VESSEL, 464
AttachmentCount
 VESSEL, 463
ATTACHMENTHANDLE
 handle, 18
BaseInterface
 oapiGetBaseEquPos, 96
 oapiGetBasePadCount, 96
 oapiGetBasePadEquPos, 97
 oapiGetBasePadNav, 97
 oapiGetBasePadStatus, 97
 oapiGetBasePlanet, 98
BASELINE
 oapi::Sketchpad, 318
BEACONLIGHTSPEC, 197
bEphemeris
 CELBODY, 201
Bit flags for blitting operations, 15
Bit flags for planetarium mode elements, 14
Bitflags for EXHAUSTSPEC flags field., 31
BK_OPAQUE
 oapi::Sketchpad, 319
BK_TRANSPARENT
 oapi::Sketchpad, 319
BkgMode
 oapi::Sketchpad, 318
BLT_TGTCOLORKEY
 bltflag, 16
bltflag
 BLT_TGTCOLORKEY, 16
Body functions, 61
BOLD
 oapi::Font, 219
BOTTOM
 oapi::Sketchpad, 318
Brush
 oapi::Brush, 199
ButtonLabel
 MFD, 284
ButtonMenu
 MFD, 284
Camera
 oapiCameraAperture, 84
 oapiCameraAttach, 84
 oapiCameraAzimuth, 85
 oapiCameraGlobalDir, 85
 oapiCameraGlobalPos, 85
 oapiCameraInternal, 85
 oapiCameraMode, 85
 oapiCameraPolar, 86
 oapiCameraRotAzimuth, 86
 oapiCameraRotPolar, 86
 oapiCameraScaleDist, 86
 oapiCameraSetAperture, 87
 oapiCameraSetCockpitDir, 87
 oapiCameraTarget, 87
 oapiCameraTargetDist, 88
 oapiCockpitMode, 88
 Camera functions, 83
CELBODY, 200
 bEphemeris, 201
 clbkAtmParam, 203
 clbkEphemeris, 201
 clbkFastEphemeris, 202
 clbkInit, 201
 Version, 201
CELBODY2, 204
 ~CELBODY2, 206
 CELBODY2, 206
 clbkInit, 206
 FreeAtmosphere, 208
 FreeAtmosphereModule, 209
 GetAtmosphere, 207
 GetChild, 206
 GetParent, 206
 LegacyAtmosphereInterface, 207
 LoadAtmosphereModule, 208
 SetAtmosphere, 207
 SidRotPeriod, 207
CENTER
 oapi::Sketchpad, 318
cfgprm
 CFGPRM_AMBIENTLEVEL, 11
 CFGPRM_ATMFOG, 11
 CFGPRM_ATMHAZE, 11
 CFGPRM_CLOUDS, 11
 CFGPRM_CLOUDSHADOWS, 11
 CFGPRM_CSPHEREINTENS, 11
 CFGPRM_CSPHERETEXTURE, 12
 CFGPRM_LOCALLIGHT, 12
 CFGPRM_MAXLIGHT, 12
 CFGPRM_OBJECTSHADOWS, 12
 CFGPRM_OBJECTSPECULAR, 12
 CFGPRM_PLANETARIUMFLAG, 12
 CFGPRM_STARRENDERPRM, 12
 CFGPRM_SURFACELIGHTBRT, 13
 CFGPRM_SURFACELIGHTS, 13

CFGPRM_SURFACEMAXLEVEL, 13
CFGPRM_SURFACEPATCHAP, 13
CFGPRM_SURFACEREFLECT, 13
CFGPRM_SURFACERIPPLE, 13
CFGPRM_SURFACESPECULAR, 13
CFGPRM_VESSELSHADOWS, 14
CFGPRM_AMBIENTLEVEL
 cfgprm, 11
CFGPRM_ATMFOG
 cfgprm, 11
CFGPRM_ATMHAZE
 cfgprm, 11
CFGPRM_CLOUDS
 cfgprm, 11
CFGPRM_CLOUDSHADOWS
 cfgprm, 11
CFGPRM_CSHEREINTENS
 cfgprm, 11
CFGPRM_CSHERETEXTURE
 cfgprm, 12
CFGPRM_LOCALLIGHT
 cfgprm, 12
CFGPRM_MAXLIGHT
 cfgprm, 12
CFGPRM_OBJECTSHADOWS
 cfgprm, 12
CFGPRM_OBJECTSPECULAR
 cfgprm, 12
CFGPRM_PLANETARIUMFLAG
 cfgprm, 12
CFGPRM_STARRENDERPRM
 cfgprm, 12
CFGPRM_SURFACELIGHTBRT
 cfgprm, 13
CFGPRM_SURFACELIGHTS
 cfgprm, 13
CFGPRM_SURFACEMAXLEVEL
 cfgprm, 13
CFGPRM_SURFACEPATCHAP
 cfgprm, 13
CFGPRM_SURFACEREFLECT
 cfgprm, 13
CFGPRM_SURFACERIPPLE
 cfgprm, 13
CFGPRM_SURFACESPECULAR
 cfgprm, 13
CFGPRM_VESSELSHADOWS
 cfgprm, 14
clbkADCtrlMode
 VESSEL2, 491
clbkAnimate
 VESSEL2, 493
clbkAtmParam
 CELBODY, 203
clbkBlt
 oapi::GraphicsClient, 243, 244
clbkCloseSession
 oapi::GraphicsClient, 251
clbkConstants
 ATMOSPHERE, 195
clbkConsumeBufferedKey
 VESSEL2, 494
clbkConsumeDirectKey
 VESSEL2, 494
clbkCopyBitmap
 oapi::GraphicsClient, 246
clbkCreateAnnotation
 oapi::GraphicsClient, 234
clbkCreateBrush
 oapi::GraphicsClient, 249
clbkCreateExhaustStream
 oapi::GraphicsClient, 233
clbkCreateFont
 oapi::GraphicsClient, 247
clbkCreateParticleStream
 oapi::GraphicsClient, 232
clbkCreatePen
 oapi::GraphicsClient, 248
clbkCreateReentryStream
 oapi::GraphicsClient, 234
clbkCreateRenderWindow
 oapi::GraphicsClient, 251
clbkCreateSurface
 oapi::GraphicsClient, 240, 241
clbkCreateTexture
 oapi::GraphicsClient, 240
clbkDeleteVessel
 oapi::Module, 295
clbkDestroyRenderWindow
 oapi::GraphicsClient, 252
clbkDisplayFrame
 oapi::GraphicsClient, 253
clbkDockEvent
 VESSEL2, 493
clbkDrawHUD
 VESSEL2, 490
 VESSEL3, 506
clbkEditMeshGroup
 oapi::GraphicsClient, 232
clbkEphemeris
 CELBODY, 201
clbkFastEphemeris
 CELBODY, 202
clbkFillSurface
 oapi::GraphicsClient, 245, 246
clbkFocusChanged
 ExternMFD, 217
 oapi::Module, 294

VESSEL2, 487
clbkFullscreenMode
 oapi::GraphicsClient, 236
clbkGeneric
 VESSEL3, 505
clbkGetDeviceColour
 oapi::GraphicsClient, 243
clbkGetMesh
 oapi::GraphicsClient, 231
clbkGetRadiationForce
 VESSEL3, 508
clbkGetRenderParam
 oapi::GraphicsClient, 236
clbkGetSketchpad
 oapi::GraphicsClient, 247
clbkGetSurfaceDC
 oapi::GraphicsClient, 250
clbkGetSurfaceSize
 oapi::GraphicsClient, 242
clbkGetViewportSize
 oapi::GraphicsClient, 236
clbkHUDMode
 VESSEL2, 492
clbkIncrSurfaceRef
 oapi::GraphicsClient, 241
clbkInit
 CELBODY, 201
 CELBODY2, 206
clbkInitialise
 oapi::GraphicsClient, 227
clbkLoadGenericCockpit
 VESSEL2, 495
clbkLoadPanel
 VESSEL2, 495
clbkLoadPanel2D
 VESSEL3, 505
clbkLoadStateEx
 VESSEL2, 486
clbkLoadTexture
 oapi::GraphicsClient, 228
clbkLoadVC
 VESSEL2, 497
clbkMFDMode
 VESSEL2, 492
clbkName
 ATMOSPHERE, 194
clbkNavMode
 VESSEL2, 493
clbkNewVessel
 oapi::Module, 295
clbkOpen
 LaunchpadItem, 266
clbkPanelMouseEvent
 VESSEL2, 496
 VESSEL3, 504
clbkPanelRedrawEvent
 VESSEL2, 496
 VESSEL3, 504
clbkParams
 ATMOSPHERE, 195
clbkPause
 oapi::Module, 296
clbkPlaybackEvent
 VESSEL2, 489
clbkPostCreation
 oapi::GraphicsClient, 251
 VESSEL2, 487
clbkPostStep
 oapi::Module, 293
 VESSEL2, 488
clbkPreOpenPopup
 oapi::GraphicsClient, 232
clbkPreStep
 oapi::Module, 293
 VESSEL2, 488
clbkRCSMode
 VESSEL2, 491
clbkRefreshButtons
 ExternMFD, 217
clbkRefreshDisplay
 ExternMFD, 217
clbkRefreshVideoData
 oapi::GraphicsClient, 227
clbkReleaseBrush
 oapi::GraphicsClient, 249
clbkReleaseFont
 oapi::GraphicsClient, 248
clbkReleasePen
 oapi::GraphicsClient, 249
clbkReleaseSketchpad
 oapi::GraphicsClient, 247
clbkReleaseSurface
 oapi::GraphicsClient, 241
clbkReleaseSurfaceDC
 oapi::GraphicsClient, 250
clbkReleaseTexture
 oapi::GraphicsClient, 228
clbkRender2DPanel
 oapi::GraphicsClient, 239
clbkRenderHUD
 VESSEL3, 507
clbkRenderScene
 oapi::GraphicsClient, 253
clbkSaveState
 VESSEL2, 486
clbkScaleBlt
 oapi::GraphicsClient, 245
clbkSetClassCaps

VESSEL2, 485
clbkSetMeshMaterial
 oapi::GraphicsClient, 229
clbkSetMeshProperty
 oapi::GraphicsClient, 229
clbkSetMeshTexture
 oapi::GraphicsClient, 229
clbkSetStateEx
 VESSEL2, 487
clbkSetSurfaceColourKey
 oapi::GraphicsClient, 242
clbkSimulationEnd
 oapi::Module, 293
clbkSimulationStart
 oapi::Module, 293
clbkStoreMeshPersistent
 oapi::GraphicsClient, 254
clbkTimeAccChanged
 oapi::Module, 295
clbkTimeJump
 oapi::Module, 294
clbkUpdate
 ExternMFD, 217
 oapi::GraphicsClient, 252
clbkUseLaunchpadVideoTab
 oapi::GraphicsClient, 251
clbkVCMouseEvent
 VESSEL2, 498
clbkVCRedrawEvent
 VESSEL2, 498
clbkVesselJump
 oapi::Module, 295
clbkVisEvent
 oapi::GraphicsClient, 231
clbkVisualCreated
 VESSEL2, 489
clbkVisualDestroyed
 VESSEL2, 490
clbkWriteConfig
 LaunchpadItem, 266
ClearAirfoilDefinitions
 VESSEL, 388
ClearAttachments
 VESSEL, 462
ClearBeacons
 VESSEL, 474
ClearControlSurfaceDefinitions
 VESSEL, 391
ClearDockDefinitions
 VESSEL, 457
ClearLightEmitters
 VESSEL, 477
ClearMeshes
 VESSEL, 439, 480
ClearPropellantResources
 VESSEL, 403
ClearThrusterDefinitions
 VESSEL, 410
ClearVariableDragElements
 VESSEL, 393
COLOUR4, 209
Configuration parameter identifiers, 10
ConsumeButton
 MFD, 283
ConsumeKeyBuffered
 MFD, 283
ConsumeKeyImmediate
 MFD, 283
Control surface axis orientation, 35
Coordinate transformations, 79
CopyMeshFromTemplate
 VESSEL, 445
Create
 VESSEL, 482
CreateAirfoil
 VESSEL, 384
CreateAirfoil2
 VESSEL, 385
CreateAirfoil3
 VESSEL, 386
CreateAnimation
 VESSEL, 447
CreateAttachment
 VESSEL, 461
CreateControlSurface
 VESSEL, 388
CreateControlSurface2
 VESSEL, 389
CreateControlSurface3
 VESSEL, 390
CreateDock
 VESSEL, 456
CreatePropellantResource
 VESSEL, 402
CreateThruster
 VESSEL, 408
CreateThrusterGroup
 VESSEL, 419
CreateVariableDragElement
 VESSEL, 392
crossp
 vec, 23
CTRLSURFHANDLE
 handle, 18
Custom MFD mode definition, 142
Customisation - custom menu, dialogs, 149
CustomMFD
 oapiDisableMFDMode, 143

oapiGetMFDModeSpecEx, 143
oapiRegisterMFDMode, 144
oapiUnregisterMFDMode, 144

DeactivateNavmode
 VESSEL, 374

Defines and Enumerations, 17

DefSetState
 VESSEL, 367

DefSetStateEx
 VESSEL, 368

DelAirfoil
 VESSEL, 388

DelAnimation
 VESSEL, 448

DelAnimationComponent
 VESSEL, 449

DelAttachment
 VESSEL, 461

DelBeacon
 VESSEL, 474

DelControlSurface
 VESSEL, 390

DelDock
 VESSEL, 456

DelExhaust
 VESSEL, 468

DelExhaustStream
 VESSEL, 472

DelLightEmitter
 VESSEL, 476

DelMesh
 VESSEL, 441

DelPropellantResource
 VESSEL, 403

DelThruster
 VESSEL, 409

DelThrusterGroup
 VESSEL, 420, 478

Description
 LaunchpadItem, 265

Detach
 oapi::ParticleStream, 305

DetachChild
 VESSEL, 465

DEVMESHHANDLE
 handle, 19

Dialog
 oapiAddTitleButton, 150
 oapiCloseDialog, 150
 oapiDefDialogProc, 151
 oapiFindDialog, 151
 oapiFindLaunchpadItem, 152
 oapiGetDialogContext, 152

oapiOpenDialog, 152
oapiOpenDialogEx, 153
oapiOpenHelp, 154
oapiOpenLaunchpadHelp, 154
oapiRegisterCustomCmd, 154
oapiRegisterLaunchpadItem, 155
oapiUnregisterCustomCmd, 155
oapiUnregisterLaunchpadItem, 155

DIFFUSE
 PARTICLESTREAMSPEC, 309

dist
 vec, 23

Dock
 VESSEL, 459

DockCount
 VESSEL, 458

DOCKHANDLE
 handle, 19

DockingStatus
 VESSEL, 459

dotp
 vec, 23

Drawing support functions, 133

DrawSupport
 oapiCreateBrush, 134
 oapiCreateFont, 134
 oapiCreatePen, 135
 oapiGetDC, 135
 oapiGetSketchpad, 136
 oapiReleaseBrush, 136
 oapiReleaseDC, 137
 oapiReleaseFont, 137
 oapiReleasePen, 137
 oapiReleaseSketchpad, 137

EditAirfoil
 VESSEL, 387

ELEMENTS, 210

Ellipse
 oapi::Sketchpad, 325

EMISSIVE
 PARTICLESTREAMSPEC, 309

EnableIDS
 VESSEL, 433

EnableTransponder
 VESSEL, 432

ENGINE_ATTITUDE
 thrusterparam, 33

ENGINE_HOVER
 thrusterparam, 33

ENGINE_MAIN
 thrusterparam, 33

ENGINE_RETRO
 thrusterparam, 33

ENGINESTATUS, 211
ENGINETYPE
 thrusterparam, 33
Ephemeris data format bitflags, 10
EXHAUSTSPEC, 212
ExitModule
 general_clbk, 184
ExternMFD, 213
 ~ExternMFD, 215
 Active, 215
 clbkFocusChanged, 217
 clbkRefreshButtons, 217
 clbkRefreshDisplay, 217
 clbkUpdate, 217
 ExternMFD, 215
 GetButtonLabel, 216
 GetDisplaySurface, 216
 GetVessel, 215
 Id, 215
 OpenModeHelp, 217
 ProcessButton, 216
 Resize, 217
 SendKey, 217
 SetMode, 217
 SetVessel, 216
File IO Functions, 156
FILEHANDLE
 handle, 19
FileIO
 oapiCloseFile, 157
 oapiOpenFile, 157
 oapiReadItem_bool, 158
 oapiReadItem_float, 159
 oapiReadItem_int, 159
 oapiReadItem_string, 159
 oapiReadItem_vec, 160
 oapiReadScenario_nextline, 160
 oapiSaveScenario, 161
 oapiWriteItem_bool, 161
 oapiWriteItem_float, 161
 oapiWriteItem_int, 161
 oapiWriteItem_string, 162
 oapiWriteItem_vec, 162
 oapiWriteLine, 162
 oapiWriteLog, 163
 oapiWriteScenario_float, 163
 oapiWriteScenario_int, 163
 oapiWriteScenario_string, 163
 oapiWriteScenario_vec, 163
FindRange
 GraphMFD, 261
flag
 VESSELSTATUS, 510
 VESSELSTATUS2, 513
FogParam, 217
Font
 oapi::Font, 219
FreeAtmosphere
 CELBODY2, 208
FreeAtmosphereModule
 CELBODY2, 209
Functions for planetary bodies, 88
General module callback functions, 183
general_clbk
 ExitModule, 184
 InitModule, 184
Generic vessel message identifiers, 42
GetADCctrlMode
 VESSEL, 373
GetAirfoilParam
 VESSEL, 386
GetAirspeed
 VESSEL, 383
GetAltitude
 VESSEL, 379
GetAngularAcc
 VESSEL, 370
GetAngularMoment
 VESSEL, 371
GetAngularVel
 VESSEL, 370
GetAnimPtr
 VESSEL, 450
GetAOA
 VESSEL, 384
GetApDist
 VESSEL, 379
GetArgPer
 VESSEL, 378
GetAtmDensity
 VESSEL, 381
GetAtmosphere
 CELBODY2, 207
GetAtmPressure
 VESSEL, 382
GetAtmRef
 VESSEL, 381
GetAtmTemperature
 VESSEL, 381
GetAttachmentHandle
 VESSEL, 464
GetAttachmentId
 VESSEL, 463
GetAttachmentIndex
 VESSEL, 464
GetAttachmentParams

VESSEL, 462
GetAttachmentStatus
 VESSEL, 463
GetAttenuation
 PointLight, 313
GetAttitudeLinLevel
 VESSEL, 429
GetAttitudeMode
 VESSEL, 427
GetAttitudeRotLevel
 VESSEL, 428
GetBank
 VESSEL, 380
GetBankMomentScale
 VESSEL, 479
GetBaseShadowGeometry
 oapi::GraphicsClient, 239
GetBaseStructures
 oapi::GraphicsClient, 239
GetBaseTileList
 oapi::GraphicsClient, 238
GetBeacon
 VESSEL, 475
GetButtonLabel
 ExternMFD, 216
GetCameraDefaultDirection
 VESSEL, 436
GetCameraOffset
 VESSEL, 435
GetCelestialMarkers
 oapi::GraphicsClient, 255
GetCharSize
 oapi::Sketchpad, 321
GetChild
 CELBODY2, 206
GetClassName
 VESSEL, 357
GetClipRadius
 VESSEL, 360
GetCOG_elev
 VESSEL, 362
GetConfigParam
 oapi::GraphicsClient, 237
GetControlSurfaceLevel
 VESSEL, 392
GetCrossSections
 VESSEL, 364
GetCW
 VESSEL, 393
GetDamageModel
 VESSEL, 358
GetDC
 oapi::Sketchpad, 328
GetDefaultColour

 MFD2, 290
GetDefaultFont
 MFD2, 289
GetDefaultPen
 MFD2, 289
GetDefaultPropellantResource
 VESSEL, 407
GetDevMesh
 VESSEL, 444
GetDirection
 LightEmitter, 272
GetDirectionRef
 LightEmitter, 273
GetDisplaySurface
 ExternMFD, 216
GetDockHandle
 VESSEL, 458
GetDockParams
 VESSEL, 458
GetDockStatus
 VESSEL, 458
GetDrag
 VESSEL, 399
GetDragVector
 VESSEL, 400
GetDynPressure
 VESSEL, 382
GetEditorModule
 VESSEL, 357
GetElements
 VESSEL, 375, 376
GetEmptyMass
 VESSEL, 362
GetEnableFocus
 VESSEL, 358
GetEquPos
 VESSEL, 381
GetExhaustCount
 VESSEL, 468
GetExhaustLevel
 VESSEL, 469
GetExhaustSpec
 VESSEL, 468, 469
GetFlightModel
 VESSEL, 358
GetFlightStatus
 VESSEL, 368
GetForceVector
 VESSEL, 401
GetFuelMass
 VESSEL, 408
GetFuelRate
 VESSEL, 408
GetGDIFont

oapi::Font, 220
GetGlobalOrientation
 VESSEL, 371
GetGlobalPos
 VESSEL, 369
GetGlobalVel
 VESSEL, 369
GetGravityGradientDamping
 VESSEL, 366
GetGravityRef
 VESSEL, 375
GetGroupThruster
 VESSEL, 422, 423
GetGroupThrusterCount
 VESSEL, 422
GetHandle
 VESSEL, 357
GetHeight
 MFD2, 288
GetHorizonAirspeedVector
 VESSEL, 383
GetIDS
 VESSEL, 434
GetISP
 VESSEL, 418
GetLift
 VESSEL, 399
GetLiftVector
 VESSEL, 400
GetLightEmitter
 VESSEL, 476
GetLinearMoment
 VESSEL, 370
GetMachNumber
 VESSEL, 382
GetManualControlLevel
 VESSEL, 426
GetMass
 VESSEL, 368
GetMaxFuelMass
 VESSEL, 407
GetMesh
 VESSEL, 443
GetMeshCount
 VESSEL, 443
GetMeshName
 VESSEL, 444
GetMeshOffset
 VESSEL, 443
GetMeshTemplate
 VESSEL, 444
GetMeshVisibilityMode
 VESSEL, 445
GetMFDSurface
 oapi::GraphicsClient, 238
GetModule
 oapi::ModuleNV, 297
GetName
 VESSEL, 357
GetNavChannel
 VESSEL, 432
GetNavCount
 VESSEL, 431
GetNavmodeState
 VESSEL, 375
GetNavRecv
 VESSEL, 480
GetNavRecvFreq
 VESSEL, 432
GetNavSource
 VESSEL, 435
GetNosewheelSteering
 VESSEL, 472
GetParent
 CELBODY2, 206
GetPeDist
 VESSEL, 378
GetPenumbra
 SpotLight, 331
GetPitch
 VESSEL, 380
GetPitchMomentScale
 VESSEL, 396
GetPMI
 VESSEL, 365
GetPopupList
 oapi::GraphicsClient, 236
GetPosition
 LightEmitter, 270
GetPositionRef
 LightEmitter, 271
GetPropellantCount
 VESSEL, 403
GetPropellantEfficiency
 VESSEL, 405
GetPropellantFlowrate
 VESSEL, 406
GetPropellantHandleByIndex
 VESSEL, 403
GetPropellantMass
 VESSEL, 404
GetPropellantMaxMass
 VESSEL, 404
GetRange
 PointLight, 312
GetRelativePos
 VESSEL, 369
GetRelativeVel

VESSEL, 370
GetRotationMatrix
 VESSEL, 453
GetRotDrag
 VESSEL, 395
GetShipAirspeedVector
 VESSEL, 383
GetSimMJD
 oapi::ModuleNV, 298
GetSimStep
 oapi::ModuleNV, 298
GetSimTime
 oapi::ModuleNV, 298
GetSize
 VESSEL, 359
GetSlipAngle
 VESSEL, 384
GetSMi
 VESSEL, 378
GetStatus
 VESSEL, 367
GetStatusEx
 VESSEL, 367
GetSuperstructureCG
 VESSEL, 452
GetSurface
 oapi::Sketchpad, 328
GetSurfaceMarkers
 oapi::GraphicsClient, 255
GetSurfaceRef
 VESSEL, 379
GetTextWidth
 oapi::Sketchpad, 322
GetThrusterCount
 VESSEL, 410
GetThrusterDir
 VESSEL, 412
GetThrusterGroupHandle
 VESSEL, 421
GetThrusterGroupLevel
 VESSEL, 426
GetThrusterHandleByIndex
 VESSEL, 410
GetThrusterIsp
 VESSEL, 415
GetThrusterIsp0
 VESSEL, 414
GetThrusterLevel
 VESSEL, 416
GetThrusterMax
 VESSEL, 413, 414
GetThrusterMax0
 VESSEL, 412
GetThrusterMoment
 VESSEL, 418
GetThrusterRef
 VESSEL, 411
GetThrusterResource
 VESSEL, 410
GetThrustVector
 VESSEL, 400
GetTorqueVector
 VESSEL, 401
GetTotalPropellantFlowrate
 VESSEL, 406
GetTotalPropellantMass
 VESSEL, 405
GetTouchdownPoints
 VESSEL, 363
GetTransponder
 VESSEL, 434
GetTrimScale
 VESSEL, 397
GetUmbra
 SpotLight, 331
 GetUserThrusterGroupCount
 VESSEL, 423
 GetUserThrusterGroupHandleByIndex
 VESSEL, 421
GetVCHUDSurface
 oapi::GraphicsClient, 238
GetVCMFDSurface
 oapi::GraphicsClient, 238
GetVessel
 ExternMFD, 215
GetVideoData
 oapi::GraphicsClient, 235
GetWeightVector
 VESSEL, 399
GetWheelbrakeLevel
 VESSEL, 473
GetWidth
 MFD2, 288
GetWingAspect
 VESSEL, 394
GetWingEffectiveness
 VESSEL, 395
GetYaw
 VESSEL, 380
GetYawMomentScale
 VESSEL, 397
Global2Local
 VESSEL, 455
GlobalRot
 VESSEL, 453
GraphicsClient
 oapi::GraphicsClient, 227
GraphMFD, 258

AddGraph, 259
AddPlot, 259
FindRange, 261
GraphMFD, 259
Plot, 261
SetAutoRange, 260
SetAutoTicks, 260
SetAxisTitle, 261
SetRange, 260
GroundContact
 VESSEL, 372
GROUPEDITSPEC, 262

handle
 AIRFOILHANDLE, 18
 ANIMATIONCOMPONENT_HANDLE, 18
 ATTACHMENTHANDLE, 18
 CTRLSURFHANDLE, 18
 DEVMESHHANDLE, 19
 DOCKHANDLE, 19
 FILEHANDLE, 19
 INTERPRETERHANDLE, 19
 LAUNCHPADITEM_HANDLE, 19
 MESHHANDLE, 19
 NAVHANDLE, 19
 NOTEHANDLE, 19
 OBJHANDLE, 19
 PANELHANDLE, 19
 PROPELLANT_HANDLE, 19
 PSTREAM_HANDLE, 20
 SURFHANDLE, 20
 THGROUP_HANDLE, 20
 THRUSTER_HANDLE, 20
 VISHANDLE, 20
Handles, 18
HELPCONTEXT, 263
HorizonInvRot
 VESSEL, 454
HorizonRot
 VESSEL, 454
HUD mode identifiers, 38
HUD, MFD and panel functions, 120
HUDPARAM, 263

Id
 ExternMFD, 215
Identifiers for visual events, 35
IncEngineLevel
 VESSEL, 478
IncThrusterGroupLevel
 VESSEL, 424
IncThrusterGroupLevel_SingleStep
 VESSEL, 425
IncThrusterLevel
 VESSEL, 417
 IncThrusterLevel_SingleStep
 VESSEL, 418
InitModule
 general_clbk, 184
InitNavRadios
 VESSEL, 431
InsertMesh
 VESSEL, 440, 441
INTERPRETERHANDLE
 handle, 19
InvalidateButtons
 MFD, 281
InvalidateDisplay
 MFD, 281
IsActive
 LightEmitter, 270
ITALIC
 oapi::Font, 219

Keyboard key identifiers, 170

LaunchpadItem, 264
 clbkOpen, 266
 clbkWriteConfig, 266
 Description, 265
 Name, 265
 OpenDialog, 265
LAUNCHPADITEM_HANDLE
 handle, 19
LaunchpadVideoTab
 oapi::GraphicsClient, 254
LaunchpadVideoWndProc
 oapi::GraphicsClient, 235
LEFT
 oapi::Sketchpad, 318
LegacyAtmosphereInterface
 CELBODY2, 207
length
 vec, 23
Level
 oapi::ParticleStream, 306
LEVELMAP
 PARTICLESTREAMSPEC, 309
levelmap
 PARTICLESTREAMSPEC, 309
LIFT_HORIZONTAL
 airfoilliftdir, 34
LIFT_VERTICAL
 airfoilliftdir, 34
Light beacon shape parameters, 31
LightEmitter, 267
 Activate, 270
 GetDirection, 272

GetDirectionRef, 273
GetPosition, 270
GetPositionRef, 271
IsActive, 270
LightEmitter, 269
SetDirection, 272
SetDirectionRef, 272
SetPosition, 270
SetPositionRef, 271
ShiftExplicitPosition, 271
LightEmitterCount
 VESSEL, 476
Line
 oapi::Sketchpad, 324
LineTo
 oapi::Sketchpad, 324
Listclbkflag, 32
LISTENTRY, 273
Listentryflag, 32
LoadAtmosphereModule
 CELBODY2, 208
LoadConstellationLines
 oapi::GraphicsClient, 255
LoadMeshClbkFunc
 Mesh, 111
LoadStars
 oapi::GraphicsClient, 254
Local lighting interface, 31
Local2Global
 VESSEL, 454
Local2Rel
 VESSEL, 455
Logical key ids, 176
LTYPE
 PARTICLESTREAMSPEC, 309
ltype
 PARTICLESTREAMSPEC, 309
LVL_FLAT
 PARTICLESTREAMSPEC, 309
LVL_LIN
 PARTICLESTREAMSPEC, 309
LVL_PLIN
 PARTICLESTREAMSPEC, 309
LVL_PSQRT
 PARTICLESTREAMSPEC, 309
LVL_SQRT
 PARTICLESTREAMSPEC, 309
Manual control device identifiers, 37
Manual control mode identifiers, 37
MATERIAL, 274
MATRIX3, 274
Mesh
 LoadMeshClbkFunc, 111
oapiAddMaterial, 111
oapiCreateMesh, 111
oapiDeleteMaterial, 112
oapiDeleteMesh, 112
oapiEditMeshGroup, 112
oapiGetTextureHandle, 113
oapiLoadMesh, 113
oapiLoadMeshGlobal, 114
oapiLoadTexture, 115
oapiMeshGroup, 115
oapiMeshGroupCount, 116
oapiMeshMaterial, 116
oapiMeshMaterialCount, 117
oapiMeshTextureCount, 117
oapiObjectVisualPtr, 117
oapiParticleSetLevelRef, 118
oapiReleaseTexture, 118
oapiSetMaterial, 118
oapiSetMeshProperty, 119
oapiSetTexture, 120
Mesh group editing flags, 29
MESHGROUP, 275
MESHGROUP_TRANSFORM, 276
MESHGROUPEX, 277
MeshgroupTransform
 VESSEL, 446
MESHHANDLE
 handle, 19
MeshModified
 VESSEL, 446
MFD, 278
 ButtonLabel, 284
 ButtonMenu, 284
 ConsumeButton, 283
 ConsumeKeyBuffered, 283
 ConsumeKeyImmediate, 283
 InvalidateButtons, 281
 InvalidateDisplay, 281
 MFD, 280
 ReadStatus, 285
 RecallStatus, 286
 SelectDefaultFont, 282
 SelectDefaultPen, 282
 StoreStatus, 285
 Title, 281
 Update, 281
 WriteStatus, 285
MFD identifiers, 40
MFD mode identifiers, 39
MFD2, 286
 GetDefaultColour, 290
 GetDefaultFont, 289
 GetDefaultPen, 289
 GetHeight, 288

GetWidth, 288
MFD2, 287
Title, 288
Update, 288
Module
 oapi::Module, 292
ModuleNV
 oapi::ModuleNV, 297
Mouse event identifiers, 41
MoveTo
 oapi::Sketchpad, 324
mul
 vec, 24
Name
 LaunchpadItem, 265
NAVDATA, 299
NAVHANDLE
 handle, 19
Navigation mode identifiers, 36
Navigation radio transmitter functions, 103
Navigation radio transmitter types, 43
NavRadio
 oapiGetNavChannel, 104
 oapiGetNavData, 104
 oapiGetNavDescr, 104
 oapiGetNavFreq, 105
 oapiGetNavPos, 105
 oapiGetNavRange, 105
 oapiGetNavSignal, 106
 oapiGetNavType, 106
 oapiNavInRange, 107
NonsphericalGravityEnabled
 VESSEL, 373
NORMAL
 oapi::Font, 219
normalise
 vec, 24
NOTEHANDLE
 handle, 19
NTVERTEX, 300

oapi
 oapiDebugString, 46
 oapiGetBarycentre, 46
 oapiGetCmdLine, 47
 oapiGetInducedDrag, 47
 oapiGetModuleVersion, 48
 oapiGetOrbiterInstance, 48
 oapiGetOrbiterVersion, 48
 oapiGetPanelScale, 49
 oapiGetViewportSize, 49
 oapiGetWaveDrag, 49
 oapiRegisterExhaustTexture, 50
 oapiRegisterGraphicsClient, 50
 oapiRegisterModule, 51
 oapiRegisterReentryTexture, 51
oapi::Brush, 199
 Brush, 199
oapi::DrawingTool, 210
oapi::Font, 218
 BOLD, 219
 Font, 219
 GetGDIFont, 220
 ITALIC, 219
 NORMAL, 219
 Style, 219
 UNDERLINE, 219
oapi::GraphicsClient, 220
 ~GraphicsClient, 227
clbkBlt, 243, 244
clbkCloseSession, 251
clbkCopyBitmap, 246
clbkCreateAnnotation, 234
clbkCreateBrush, 249
clbkCreateExhaustStream, 233
clbkCreateFont, 247
clbkCreateParticleStream, 232
clbkCreatePen, 248
clbkCreateReentryStream, 234
clbkCreateRenderWindow, 251
clbkCreateSurface, 240, 241
clbkCreateTexture, 240
clbkDestroyRenderWindow, 252
clbkDisplayFrame, 253
clbkEditMeshGroup, 232
clbkFillSurface, 245, 246
clbkFullscreenMode, 236
clbkGetDeviceColour, 243
clbkGetMesh, 231
clbkGetRenderParam, 236
clbkGetSketchpad, 247
clbkGetSurfaceDC, 250
clbkGetSurfaceSize, 242
clbkGetViewportSize, 236
clbkIncrSurfaceRef, 241
clbkInitialise, 227
clbkLoadTexture, 228
clbkPostCreation, 251
clbkPreOpenPopup, 232
clbkRefreshVideoData, 227
clbkReleaseBrush, 249
clbkReleaseFont, 248
clbkReleasePen, 249
clbkReleaseSketchpad, 247
clbkReleaseSurface, 241
clbkReleaseSurfaceDC, 250
clbkReleaseTexture, 228

clbkRender2DPanel, 239
clbkRenderScene, 253
clbkScaleBlt, 245
clbkSetMeshMaterial, 229
clbkSetMeshProperty, 229
clbkSetMeshTexture, 229
clbkSetSurfaceColourKey, 242
clbkStoreMeshPersistent, 254
clbkUpdate, 252
clbkUseLaunchpadVideoTab, 251
clbkVisEvent, 231
GetBaseShadowGeometry, 239
GetBaseStructures, 239
GetBaseTileList, 238
GetCelestialMarkers, 255
GetConfigParam, 237
GetMFDSurface, 238
GetPopupList, 236
GetSurfaceMarkers, 255
GetVCHUDSurface, 238
GetVCMFDSurface, 238
GetVideoData, 235
GraphicsClient, 227
LaunchpadVideoTab, 254
LaunchpadVideoWndProc, 235
LoadConstellationLines, 255
LoadStars, 254
RegisterVisObject, 230
Render2DOverlay, 253
RenderWndProc, 235
TexturePath, 237
UnregisterVisObject, 230
oapi::GraphicsClient::LABELLIST, 256
oapi::GraphicsClient::VIDEODATA, 257
oapi::IVECTOR2, 264
oapi::Module, 291
 clbkDeleteVessel, 295
 clbkFocusChanged, 294
 clbkNewVessel, 295
 clbkPause, 296
 clbkPostStep, 293
 clbkPreStep, 293
 clbkSimulationEnd, 293
 clbkSimulationStart, 293
 clbkTimeAccChanged, 295
 clbkTimeJump, 294
 clbkVesselJump, 295
 Module, 292
 RENDER_FULLSCREEN, 292
 RENDER_NONE, 292
 RENDER_WINDOW, 292
 RenderMode, 292
oapi::ModuleNV, 296
 GetModule, 297
 GetSimMJD, 298
 GetSimStep, 298
 GetSimTime, 298
 ModuleNV, 297
 Version, 297
oapi::ParticleStream, 302
 Attach, 304, 305
 Detach, 305
 Level, 306
 ParticleStream, 304
 SetFixedDir, 305
 SetFixedPos, 305
 SetLevelPtr, 306
 SetVariableDir, 306
 SetVariablePos, 306
oapi::Pen, 309
 Pen, 310
oapi::ScreenAnnotation, 313
 ScreenAnnotation, 315
 SetColour, 315
 SetPosition, 315
 SetSize, 315
 SetText, 315
oapi::Sketchpad, 316
 BASELINE, 318
 BK_OPAQUE, 319
 BK_TRANSPARENT, 319
 BkgMode, 318
 BOTTOM, 318
 CENTER, 318
 Ellipse, 325
 GetCharSize, 321
 GetDC, 328
 GetSurface, 328
 GetTextWidth, 322
 LEFT, 318
 Line, 324
 LineTo, 324
 MoveTo, 324
 Pixel, 323
 Polygon, 326
 Polyline, 326
 PolyPolygon, 327
 PolyPolyline, 327
 Rectangle, 325
 RIGHT, 318
 SetBackgroundColor, 320
 SetBackgroundMode, 321
 SetBrush, 320
 SetFont, 319
 SetOrigin, 322
 SetPen, 319
 SetTextAlign, 320
 SetTextColor, 320

Sketchpad, 319
TAlign_horizontal, 318
TAlign_vertical, 318
Text, 322
TextBox, 323
TOP, 318
oapi_body
 oapiGetGlobalPos, 62
 oapiGetGlobalVel, 62
 oapiGetMass, 62
 oapiGetRelativePos, 63
 oapiGetRelativeVel, 63
 oapiGetSize, 63
oapi_time
 oapiGetFrameRate, 99
 oapiGetPause, 99
 oapiGetSimMJD, 99
 oapiGetSimStep, 100
 oapiGetSimTime, 100
 oapiGetSysMJD, 100
 oapiGetSysStep, 100
 oapiGetSysTime, 101
 oapiGetTimeAcceleration, 101
 oapiSetPause, 101
 oapiSetSimMJD, 101
 oapiSetTimeAcceleration, 102
 oapiTime2MJD, 103
oapi_transformation
 oapiEquToGlobal, 80
 oapiEquToLocal, 80
 oapiGetRotationMatrix, 80
 oapiGlobalToEqu, 81
 oapiGlobalToLocal, 81
 oapiLocalToEqu, 82
 oapiLocalToGlobal, 82
 oapiOrthodome, 82
oapi_vessel
 oapiGetAirspeed, 66
 oapiGetAirspeedVector, 67
 oapiGetAltitude, 67
 oapiGetAtm, 67
 oapiGetAttitudeMode, 68
 oapiGetBank, 68
 oapiGetDockHandle, 69
 oapiGetDockStatus, 69
 oapiGetEmptyMass, 69
 oapiGetEngineStatus, 70
 oapiGetEquPos, 70
 oapiGetFocusAirspeed, 70
 oapiGetFocusAirspeedVector, 71
 oapiGetFocusAltitude, 71
 oapiGetFocusAttitudeMode, 71
 oapiGetFocusBank, 71
 oapiGetFocusEngineStatus, 72
 oapiGetFocusEquPos, 72
 oapiGetFocusGlobalPos, 72
 oapiGetFocusGlobalVel, 73
 oapiGetFocusHeading, 73
 oapiGetFocusPitch, 73
 oapiGetFocusRelativePos, 73
 oapiGetFocusRelativeVel, 74
 oapiGetFocusShipAirspeedVector, 74
 oapiGetFuelMass, 74
 oapiGetHeading, 75
 oapiGetMaxFuelMass, 75
 oapiGetPitch, 75
 oapiGetPropellantHandle, 76
 oapiGetPropellantMass, 76
 oapiGetPropellantMaxMass, 76
 oapiGetShipAirspeedVector, 77
 oapiSetAttitudeMode, 77
 oapiSetEmptyMass, 77
 oapiSetEngineLevel, 78
 oapiSetFocusAttitudeMode, 78
 oapiToggleAttitudeMode, 78
 oapiToggleFocusAttitudeMode, 79
oapiAddMaterial
 Mesh, 111
oapiAddTitleButton
 Dialog, 150
oapiAnnotationSetColor
 Annotations, 166
oapiAnnotationSetPos
 Annotations, 166
oapiAnnotationSetSize
 Annotations, 167
oapiAnnotationSetText
 Annotations, 167
oapiAsyncScriptCmd
 Script, 107
oapiBlt
 Surface, 139
oapiBltPanelAreaBackground
 Panel, 122
oapiCameraAperture
 Camera, 84
oapiCameraAttach
 Camera, 84
oapiCameraAzimuth
 Camera, 85
oapiCameraGlobalDir
 Camera, 85
oapiCameraGlobalPos
 Camera, 85
oapiCameraInternal
 Camera, 85
oapiCameraMode
 Camera, 85

oapiCameraPolar
 Camera, 86
oapiCameraRotAzimuth
 Camera, 86
oapiCameraRotPolar
 Camera, 86
oapiCameraScaleDist
 Camera, 86
oapiCameraSetAperture
 Camera, 87
oapiCameraSetCockpitDir
 Camera, 87
oapiCameraTarget
 Camera, 87
oapiCameraTargetDist
 Camera, 88
oapiClearSurfaceColourKey
 Surface, 140
oapiCloseDialog
 Dialog, 150
oapiCloseFile
 FileIO, 157
oapiCockpitMode
 Camera, 88
oapiColourFill
 Surface, 140
oapiCreateAnnotation
 Annotations, 167
oapiCreateBrush
 DrawSupport, 134
oapiCreateFont
 DrawSupport, 134
oapiCreateInterpreter
 Script, 108
oapiCreateMesh
 Mesh, 111
oapiCreatePen
 DrawSupport, 135
oapiCreateSurface
 Surface, 140, 141
oapiCreateTextureSurface
 Surface, 141
oapiCreateVessel
 VesselCreation, 60
oapiCreateVesselEx
 VesselCreation, 60
oapiDebugString
 oapi, 46
oapiDecHUDIntensity
 Panel, 123
oapiDefDialogProc
 Dialog, 151
oapiDelAnnotation
 Annotations, 168
oapiDeleteMaterial
 Mesh, 112
oapiDeleteMesh
 Mesh, 112
oapiDeleteVessel
 VesselCreation, 61
oapiDelInterpreter
 Script, 108
oapiDestroySurface
 Surface, 142
oapiDisableMFDMode
 CustomMFD, 143
oapiEditMeshGroup
 Mesh, 112
oapiEquToGlobal
 oapi_transformation, 80
oapiEquToLocal
 oapi_transformation, 80
oapiExecScriptCmd
 Script, 108
oapiFindDialog
 Dialog, 151
oapiFindLaunchpadItem
 Dialog, 152
oapiGetAirspeed
 oapi_vessel, 66
oapiGetAirspeedVector
 oapi_vessel, 67
oapiGetAltitude
 oapi_vessel, 67
oapiGetAtm
 oapi_vessel, 67
oapiGetAtmPressureDensity
 Obsolete, 169
oapiGetAttitudeMode
 oapi_vessel, 68
oapiGetBank
 oapi_vessel, 68
oapiGetBarycentre
 oapi, 46
oapiGetBaseByIndex
 ObjectAccess, 53
oapiGetBaseByName
 ObjectAccess, 53
oapiGetBaseCount
 ObjectAccess, 53
oapiGetBaseEquPos
 BaseInterface, 96
oapiGetBasePadCount
 BaseInterface, 96
oapiGetBasePadEquPos
 BaseInterface, 97
oapiGetBasePadNav
 BaseInterface, 97

oapiGetBasePadStatus
 BaseInterface, 97
oapiGetBasePlanet
 BaseInterface, 98
oapiGetCelbodyInterface
 ObjectAccess, 53
oapiGetCmdLine
 oapi, 47
oapiGetColour
 Utility, 164
oapiGetDC
 DrawSupport, 135
oapiGetDialogContext
 Dialog, 152
oapiGetDockHandle
 oapi_vessel, 69
oapiGetDockStatus
 oapi_vessel, 69
oapiGetEmptyMass
 oapi_vessel, 69
oapiGetEngineStatus
 oapi_vessel, 70
oapiGetEquPos
 oapi_vessel, 70
oapiGetFocusAirspeed
 oapi_vessel, 70
oapiGetFocusAirspeedVector
 oapi_vessel, 71
oapiGetFocusAltitude
 oapi_vessel, 71
oapiGetFocusAtmPressureDensity
 Obsolete, 169
oapiGetFocusAttitudeMode
 oapi_vessel, 71
oapiGetFocusBank
 oapi_vessel, 71
oapiGetFocusEngineStatus
 oapi_vessel, 72
oapiGetFocusEquPos
 oapi_vessel, 72
oapiGetFocusGlobalPos
 oapi_vessel, 72
oapiGetFocusGlobalVel
 oapi_vessel, 73
oapiGetFocusHeading
 oapi_vessel, 73
oapiGetFocusInterface
 ObjectAccess, 54
oapiGetFocusObject
 ObjectAccess, 54
oapiGetFocusPitch
 oapi_vessel, 73
oapiGetFocusRelativePos
 oapi_vessel, 73
oapiGetFocusRelativeVel
 oapi_vessel, 74
oapiGetFocusShipAirspeedVector
 oapi_vessel, 74
oapiGetFrameRate
 oapi_time, 99
oapiGetFuelMass
 oapi_vessel, 74
oapiGetGbodyByIndex
 ObjectAccess, 54
oapiGetGbodyByName
 ObjectAccess, 55
oapiGetGbodyCount
 ObjectAccess, 55
oapiGetGlobalPos
 oapi_body, 62
oapiGetGlobalVel
 oapi_body, 62
oapiGetGroundVector
 Planet, 90
oapiGetHeading
 oapi_vessel, 75
oapiGetHUDMode
 Panel, 123
oapiGetInducedDrag
 oapi, 47
oapiGetMass
 oapi_body, 62
oapiGetMaxFuelMass
 oapi_vessel, 75
oapiGetMFDMode
 Panel, 123
oapiGetMFDModeSpec
 Obsolete, 169
oapiGetMFDModeSpecEx
 CustomMFD, 143
oapiGetModuleVersion
 oapi, 48
oapiGetNavChannel
 NavRadio, 104
oapiGetNavData
 NavRadio, 104
oapiGetNavDescr
 NavRadio, 104
oapiGetNavFreq
 NavRadio, 105
oapiGetNavPos
 NavRadio, 105
oapiGetNavRange
 NavRadio, 105
oapiGetNavSignal
 NavRadio, 106
oapiGetNavType
 NavRadio, 106

oapiGetObjectByIndex
 ObjectAccess, 55
oapiGetObjectByName
 ObjectAccess, 56
oapiGetObjectCount
 ObjectAccess, 56
oapiGetObjectName
 ObjectAccess, 56
oapiGetObjectParam
 ObjectAccess, 56
oapiGetObjectType
 ObjectAccess, 57
oapiGetOrbiterInstance
 oapi, 48
oapiGetOrbiterVersion
 oapi, 48
oapiGetPanelScale
 oapi, 49
oapiGetPause
 oapi_time, 99
oapiGetPitch
 oapi_vessel, 75
oapiGetPlanetAtmConstants
 Planet, 90
oapiGetPlanetAtmParams
 Planet, 91
oapiGetPlanetCurrentRotation
 Planet, 92
oapiGetPlanetJCoeff
 Planet, 92
oapiGetPlanetJCoeffCount
 Planet, 92
oapiGetPlanetObliquity
 Planet, 93
oapiGetPlanetObliquityMatrix
 Planet, 93
oapiGetPlanetPeriod
 Planet, 94
oapiGetPlanetTheta
 Planet, 94
oapiGetPropellantHandle
 oapi_vessel, 76
oapiGetPropellantMass
 oapi_vessel, 76
oapiGetPropellantMaxMass
 oapi_vessel, 76
oapiGetRelativePos
 oapi_body, 63
oapiGetRelativeVel
 oapi_body, 63
oapiGetRotationMatrix
 oapi_transformation, 80
oapiGetShipAirspeedVector
 oapi_vessel, 77
oapiGetSimMJD
 oapi_time, 99
oapiGetSimStep
 oapi_time, 100
oapiGetSimTime
 oapi_time, 100
oapiGetSize
 oapi_body, 63
oapiGetSketchpad
 DrawSupport, 136
oapiGetStationByIndex
 Obsolete, 170
oapiGetStationByName
 Obsolete, 170
oapiGetSysMJD
 oapi_time, 100
oapiGetSysStep
 oapi_time, 100
oapiGetSysTime
 oapi_time, 101
oapiGetTextureHandle
 Mesh, 113
oapiGetTimeAcceleration
 oapi_time, 101
oapiGetVesselByIndex
 ObjectAccess, 57
oapiGetVesselByName
 ObjectAccess, 58
oapiGetVesselCount
 ObjectAccess, 58
oapiGetVesselInterface
 ObjectAccess, 58
oapiGetViewportSize
 oapi, 49
oapiGetWaveDrag
 oapi, 49
oapiGetWindVector
 Planet, 94
oapiGlobalToEqu
 oapi_transformation, 81
oapiGlobalToLocal
 oapi_transformation, 81
oapiIncHUDIntensity
 Panel, 124
oapiIsVessel
 ObjectAccess, 59
oapiLoadMesh
 Mesh, 113
oapiLoadMeshGlobal
 Mesh, 114
oapiLoadTexture
 Mesh, 115
oapiLocalToEqu
 oapi_transformation, 82

oapiLocalToGlobal
 oapi_transformation, 82
oapiMeshGroup
 Mesh, 115
oapiMeshGroupCount
 Mesh, 116
oapiMeshMaterial
 Mesh, 116
oapiMeshMaterialCount
 Mesh, 117
oapiMeshTextureCount
 Mesh, 117
oapiMFDButtonLabel
 Panel, 124
oapiNavInRange
 NavRadio, 107
oapiObjectVisualPtr
 Mesh, 117
oapiOpenDialog
 Dialog, 152
oapiOpenDialogEx
 Dialog, 153
oapiOpenFile
 FileIO, 157
oapiOpenHelp
 Dialog, 154
oapiOpenInputBox
 UserInput, 165
oapiOpenLaunchpadHelp
 Dialog, 154
oapiOpenMFD
 Panel, 124
oapiOrthodome
 oapi_transformation, 82
oapiParticleSetLevelRef
 Mesh, 118
oapiPlanetHasAtmosphere
 Planet, 95
oapiProcessMFDButton
 Panel, 125
oapiRand
 Utility, 164
oapiReadItem_bool
 FileIO, 158
oapiReadItem_float
 FileIO, 159
oapiReadItem_int
 FileIO, 159
oapiReadItem_string
 FileIO, 159
oapiReadItem_vec
 FileIO, 160
oapiReadScenario_nextline
 FileIO, 160

oapiRefreshMFDBButtons
 Panel, 125
oapiRegisterCustomCmd
 Dialog, 154
oapiRegisterExhaustTexture
 oapi, 50
oapiRegisterGraphicsClient
 oapi, 50
oapiRegisterLaunchpadItem
 Dialog, 155
oapiRegisterMFD
 Panel, 126
oapiRegisterMFDMode
 CustomMFD, 144
 Obsolete, 170
oapiRegisterModule
 oapi, 51
oapiRegisterPanelArea
 Panel, 127
oapiRegisterPanelBackground
 Panel, 128
oapiRegisterReentryTexture
 oapi, 51
oapiReleaseBrush
 DrawSupport, 136
oapiReleaseDC
 DrawSupport, 137
oapiReleaseFont
 DrawSupport, 137
oapiReleasePen
 DrawSupport, 137
oapiReleaseSketchpad
 DrawSupport, 137
oapiReleaseTexture
 Mesh, 118
oapiRenderHUD
 Panel, 128
oapiSaveScenario
 FileIO, 161
oapiSendMFDKey
 Panel, 129
oapiSetAttitudeMode
 oapi_vessel, 77
oapiSetDefNavDisplay
 Panel, 129
oapiSetDefRCSDisplay
 Panel, 129
oapiSetEmptyMass
 oapi_vessel, 77
oapiSetEngineLevel
 oapi_vessel, 78
oapiSetFocusAttitudeMode
 oapi_vessel, 78
oapiSetFocusObject

ObjectAccess, 59
oapiSetHUDMode
 Panel, 130
oapiSetMaterial
 Mesh, 118
oapiSetMeshProperty
 Mesh, 119
oapiSetPanel
 Panel, 131
oapiSetPanelNeighbours
 Panel, 131
oapiSetPause
 oapi_time, 101
oapiSetSimMJD
 oapi_time, 101
oapiSetSurfaceColourKey
 Surface, 142
oapiSetTexture
 Mesh, 120
oapiSetTimeAcceleration
 oapi_time, 102
oapiSwitchPanel
 Panel, 131
oapiTime2MJD
 oapi_time, 103
oapiToggleAttitudeMode
 oapi_vessel, 78
oapiToggleFocusAttitudeMode
 oapi_vessel, 79
oapiToggleHudColour
 Panel, 132
oapiToggleMFD_on
 Panel, 132
oapiTriggerPanelRedrawArea
 Panel, 132
oapiTriggerRedrawArea
 Panel, 132
oapiUnregisterCustomCmd
 Dialog, 155
oapiUnregisterLaunchpadItem
 Dialog, 155
oapiUnregisterMFDMode
 CustomMFD, 144
oapiVCRegisterArea
 VirtualCockpit, 145
oapiVCRegisterHUD
 VirtualCockpit, 146
oapiVCRegisterMFD
 VirtualCockpit, 146
oapiVCSetAreaClickmode_Quadrilateral
 VirtualCockpit, 147
oapiVCSetAreaClickmode_Spherical
 VirtualCockpit, 147
oapiVCSetNeighbours
 VirtualCockpit, 148
oapiVCTriggerRedrawArea
 VirtualCockpit, 148
oapiWriteItem_bool
 FileIO, 161
oapiWriteItem_float
 FileIO, 161
oapiWriteItem_int
 FileIO, 161
oapiWriteItem_string
 FileIO, 162
oapiWriteItem_vec
 FileIO, 162
oapiWriteLine
 FileIO, 162
oapiWriteLog
 FileIO, 163
oapiWriteScenario_float
 FileIO, 163
oapiWriteScenario_int
 FileIO, 163
oapiWriteScenario_string
 FileIO, 163
oapiWriteScenario_vec
 FileIO, 163
Object access functions, 51
Object parameter flags, 44
ObjectAccess
 oapiGetBaseByIndex, 53
 oapiGetBaseByName, 53
 oapiGetBaseCount, 53
 oapiGetCelbodyInterface, 53
 oapiGetFocusInterface, 54
 oapiGetFocusObject, 54
 oapiGetGbodyByIndex, 54
 oapiGetGbodyByName, 55
 oapiGetGbodyCount, 55
 oapiGetObjectByIndex, 55
 oapiGetObjectByName, 56
 oapiGetObjectCount, 56
 oapiGetObjectName, 56
 oapiGetObjectParam, 56
 oapiGetObjectType, 57
 oapiGetVesselByIndex, 57
 oapiGetVesselByName, 58
 oapiGetVesselCount, 58
 oapiGetVesselInterface, 58
 oapiIsVessel, 59
 oapiSetFocusObject, 59
OBJHANDLE
 handle, 19
Obsolete
 oapiGetAtmPressureDensity, 169
 oapiGetFocusAtmPressureDensity, 169

oapiGetMFDModeSpec, 169
oapiGetStationByIndex, 170
oapiGetStationByName, 170
oapiRegisterMFDMode, 170
Obsolete functions, 168
Onscreen annotations, 166
opcCloseRenderViewport
 plugin_clbk, 186
opcDeleteVessel
 plugin_clbk, 187
opcFocusChanged
 plugin_clbk, 187
opcOpenRenderViewport
 plugin_clbk, 188
opcPause
 plugin_clbk, 188
opcPostStep
 plugin_clbk, 188
opcPreStep
 plugin_clbk, 189
opcTimeAccChanged
 plugin_clbk, 189
OpenDialog
 LaunchpadItem, 265
OpenModeHelp
 ExternMFD, 217
operator *
 vec, 24, 25
operator *=
 vec, 25
operator+
 vec, 25, 26
operator+=
 vec, 26
operator-
 vec, 26, 27
operator-=
 vec, 27
operator/
 vec, 27
operator/=
 vec, 28
Orbiter API interface methods, 44
Orbitersdk/include/CelBodyAPI.h, 516
Orbitersdk/include/DrawAPI.h, 517
Orbitersdk/include/MFDAPI.h, 518
Orbitersdk/include/OrbiterAPI.h, 518
Orbitersdk/include/VesselAPI.h, 566
ORBITPARAM, 301
OrbitStabilised
 VESSEL, 372
outerp
 vec, 28
ovcExit
 vessel_clbk, 185
ovcInit
 vessel_clbk, 185
Panel
 oapiBltPanelAreaBackground, 122
 oapiDecHUDIntensity, 123
 oapiGetHUDMode, 123
 oapiGetMFDMode, 123
 oapiIncHUDIntensity, 124
 oapiMFDButtonLabel, 124
 oapiOpenMFD, 124
 oapiProcessMFDButton, 125
 oapiRefreshMFDButtons, 125
 oapiRegisterMFD, 126
 oapiRegisterPanelArea, 127
 oapiRegisterPanelBackground, 128
 oapiRenderHUD, 128
 oapiSendMFDKey, 129
 oapiSetDefNavDisplay, 129
 oapiSetDefRCSDisplay, 129
 oapiSetHUDMode, 130
 oapiSetPanel, 131
 oapiSetPanelNeighbours, 131
 oapiSwitchPanel, 131
 oapiToggleHUDColour, 132
 oapiToggleMFD_on, 132
 oapiTriggerPanelRedrawArea, 132
 oapiTriggerRedrawArea, 132
Panel neighbour identifiers, 40
Panel redraw event identifiers, 41
PANELHANDLE
 handle, 19
ParseScenarioLine
 VESSEL, 482
ParseScenarioLineEx
 VESSEL, 477
ParticleStream
 oapi::ParticleStream, 304
PARTICLESTREAMSPEC, 307
 DIFFUSE, 309
 EMISSIVE, 309
 LEVELMAP, 309
 levelmap, 309
 LTYPE, 309
 ltype, 309
 LVL_FLAT, 309
 LVL_LIN, 309
 LVL_PLIN, 309
 LVL_PSQRT, 309
 LVL_SQRT, 309
Pen
 oapi::Pen, 310
Pixel

oapi::Sketchpad, 323
Planet
 oapiGetGroundVector, 90
 oapiGetPlanetAtmConstants, 90
 oapiGetPlanetAtmParams, 91
 oapiGetPlanetCurrentRotation, 92
 oapiGetPlanetJCoeff, 92
 oapiGetPlanetJCoeffCount, 92
 oapiGetPlanetObliquity, 93
 oapiGetPlanetObliquityMatrix, 93
 oapiGetPlanetPeriod, 94
 oapiGetPlanetTheta, 94
 oapiGetWindVector, 94
 oapiPlanetHasAtmosphere, 95
Playback
 VESSEL, 451
Plot
 GraphMFD, 261
Plugin module callback functions, 186
plugin_clbk
 opcCloseRenderViewport, 186
 opcDeleteVessel, 187
 opcFocusChanged, 187
 opcOpenRenderViewport, 188
 opcPause, 188
 opcPostStep, 188
 opcPreStep, 189
 opcTimeAccChanged, 189
PointLight, 310
 GetAttenuation, 313
 GetRange, 312
 PointLight, 312
 SetAttenuation, 313
 SetRange, 312
Polygon
 oapi::Sketchpad, 326
Polyline
 oapi::Sketchpad, 326
PolyPolygon
 oapi::Sketchpad, 327
PolyPolyline
 oapi::Sketchpad, 327
PRM_ALT
 ATMOSPHERE, 194
PRM_AP
 ATMOSPHERE, 194
PRM_F
 ATMOSPHERE, 194
PRM_FBR
 ATMOSPHERE, 194
PRM_IN_FLAG
 ATMOSPHERE, 194
PRM_LAT
 ATMOSPHERE, 194
PRM_LNG
 ATMOSPHERE, 194
ProcessButton
 ExternMFD, 216
PROPELLANT_HANDLE
 handle, 19
PSTREAM_HANDLE
 handle, 20
RCS mode identifiers, 38
ReadStatus
 MFD, 285
RecallStatus
 MFD, 286
RecordEvent
 VESSEL, 451
Recording
 VESSEL, 450
Rectangle
 oapi::Sketchpad, 325
RegisterAnimation
 VESSEL, 447
RegisterPanelArea
 VESSEL3, 503
RegisterPanelMFDGeometry
 VESSEL3, 502
RegisterVisObject
 oapi::GraphicsClient, 230
Render parameter identifiers, 14
Render2DOOverlay
 oapi::GraphicsClient, 253
RENDER_FULLSCREEN
 oapi::Module, 292
RENDER_NONE
 oapi::Module, 292
RENDER_WINDOW
 oapi::Module, 292
RenderMode
 oapi::Module, 292
renderprm
 RP_STENCILDEPTH, 14
RenderWndProc
 oapi::GraphicsClient, 235
Resize
 ExternMFD, 217
RIGHT
 oapi::Sketchpad, 318
RP_STENCILDEPTH
 renderprm, 14
SaveDefaultState
 VESSEL, 481
ScreenAnnotation
 oapi::ScreenAnnotation, 315

Script
 oapiAsyncScriptCmd, 107
 oapiCreateInterpreter, 108
 oapiDelInterpreter, 108
 oapiExecScriptCmd, 108
Script interpreter functions, 107
SelectDefaultFont
 MFD, 282
SelectDefaultPen
 MFD, 282
SendBufferedKey
 VESSEL, 430
SendKey
 ExternMFD, 217
SetADCtrlMode
 VESSEL, 374
SetAlbedoRGB
 VESSEL, 361
SetAngularVel
 VESSEL, 371
SetAnimation
 VESSEL, 450
SetAperture
 SpotLight, 331
SetAtmosphere
 CELBODY2, 207
SetAttachmentParams
 VESSEL, 462
SetAttenuation
 PointLight, 313
SetAttitudeLinLevel
 VESSEL, 430
SetAttitudeMode
 VESSEL, 427
SetAttitudeRotLevel
 VESSEL, 429
SetAutoRange
 GraphMFD, 260
SetAutoTicks
 GraphMFD, 260
SetAxisTitle
 GraphMFD, 261
SetBackgroundColor
 oapi::Sketchpad, 320
SetBackgroundMode
 oapi::Sketchpad, 321
SetBankMomentScale
 VESSEL, 479
SetBrush
 oapi::Sketchpad, 320
SetCameraCatchAngle
 VESSEL, 437
SetCameraDefaultDirection
 VESSEL, 436
SetCameraMovement
 VESSEL, 438
SetCameraOffset
 VESSEL, 435
SetCameraRotationRange
 VESSEL, 437
SetCameraShiftRange
 VESSEL, 438
SetClipRadius
 VESSEL, 361
SetCOG_elev
 VESSEL, 480
SetColour
 oapi::ScreenAnnotation, 315
SetControlSurfaceLevel
 VESSEL, 391
SetCrossSections
 VESSEL, 364
SetCW
 VESSEL, 393
SetDefaultPropellantResource
 VESSEL, 407
SetDirection
 LightEmitter, 272
SetDirectionRef
 LightEmitter, 272
SetDockMode
 VESSEL, 460
SetDockParams
 VESSEL, 457
SetElements
 VESSEL, 377
SetEmptyMass
 VESSEL, 362
SetEnableFocus
 VESSEL, 359
SetEngineLevel
 VESSEL, 477
SetExhaustScales
 VESSEL, 478
SetFixedDir
 oapi::ParticleStream, 305
SetFixedPos
 oapi::ParticleStream, 305
SetFont
 oapi::Sketchpad, 319
SetFuelMass
 VESSEL, 408
SetGlobalOrientation
 VESSEL, 372
SetGravityGradientDamping
 VESSEL, 366
SetIDSChannel
 VESSEL, 433

SetISP
 VESSEL, 419
SetLevelPtr
 oapi::ParticleStream, 306
SetLiftCoeffFunc
 VESSEL, 398
SetMaxFuelMass
 VESSEL, 407
SetMaxWheelbrakeForce
 VESSEL, 473
SetMeshVisibilityMode
 VESSEL, 445
SetMeshVisibleInternal
 VESSEL, 481
SetMode
 ExternMFD, 217
SetNavChannel
 VESSEL, 431
SetNavRecv
 VESSEL, 480
SetNosewheelSteering
 VESSEL, 472
SetOrigin
 oapi::Sketchpad, 322
SetPanelBackground
 VESSEL3, 501
SetPanelScaling
 VESSEL3, 502
SetPen
 oapi::Sketchpad, 319
SetPitchMomentScale
 VESSEL, 396
SetPMI
 VESSEL, 365
SetPosition
 LightEmitter, 270
 oapi::ScreenAnnotation, 315
SetPositionRef
 LightEmitter, 271
SetPropellantEfficiency
 VESSEL, 406
SetPropellantMass
 VESSEL, 405
SetPropellantMaxMass
 VESSEL, 404
SetRange
 GraphMFD, 260
 PointLight, 312
SetReentryTexture
 VESSEL, 469
SetRotationMatrix
 VESSEL, 453
SetRotDrag
 VESSEL, 396
SetSize
 oapi::ScreenAnnotation, 315
 VESSEL, 360
SetSurfaceFrictionCoeff
 VESSEL, 364
SetText
 oapi::ScreenAnnotation, 315
SetTextAlign
 oapi::Sketchpad, 320
SetTextColor
 oapi::Sketchpad, 320
SetThrusterDir
 VESSEL, 412
SetThrusterGroupLevel
 VESSEL, 424
SetThrusterIsp
 VESSEL, 416
SetThrusterLevel
 VESSEL, 417
SetThrusterLevel_SingleStep
 VESSEL, 417
SetThrusterMax0
 VESSEL, 413
SetThrusterRef
 VESSEL, 411
SetThrusterResource
 VESSEL, 411
SetTouchdownPoints
 VESSEL, 363
SetTransponderChannel
 VESSEL, 433
SetTrimScale
 VESSEL, 398
SetVariableDir
 oapi::ParticleStream, 306
SetVariablePos
 oapi::ParticleStream, 306
SetVessel
 ExternMFD, 216
SetVisibilityLimit
 VESSEL, 360
SetWheelbrakeLevel
 VESSEL, 473
SetWingAspect
 VESSEL, 394
SetWingEffectiveness
 VESSEL, 395
SetYawMomentScale
 VESSEL, 397
ShiftCentreOfMass
 VESSEL, 451
ShiftCG
 VESSEL, 452
ShiftExplicitPosition

LightEmitter, 271
ShiftMesh
 VESSEL, 442
ShiftMeshes
 VESSEL, 442
SidRotPeriod
 CELBODY2, 207
Sketchpad
 oapi::Sketchpad, 319
Some useful general constants, 16
SpotLight, 328
 GetPenumbra, 331
 GetUmbra, 331
 SetAperture, 331
 SpotLight, 330
status
 VESSELSTATUS, 510
 VESSELSTATUS2, 513
StoreStatus
 MFD, 285
Structure definitions, 17
Style
 oapi::Font, 219
surf_hdg
 VESSELSTATUS2, 514
surf_lat
 VESSELSTATUS2, 514
surf_lng
 VESSELSTATUS2, 514
Surface
 oapiBlt, 139
 oapiClearSurfaceColourKey, 140
 oapiColourFill, 140
 oapiCreateSurface, 140, 141
 oapiCreateTextureSurface, 141
 oapiDestroySurface, 142
 oapiSetSurfaceColourKey, 142
Surface base interface, 95
Surface functions, 138
SURFHANDLE
 handle, 20
TAlign_horizontal
 oapi::Sketchpad, 318
TAlign_vertical
 oapi::Sketchpad, 318
Text
 oapi::Sketchpad, 322
TextBox
 oapi::Sketchpad, 323
TexturePath
 oapi::GraphicsClient, 237
THGROUP_ATT_BACK
 thrusterparam, 33
THGROUP_ATT_BANKLEFT
 thrusterparam, 33
THGROUP_ATT_BANKRIGHT
 thrusterparam, 33
THGROUP_ATT_DOWN
 thrusterparam, 33
THGROUP_ATT_FORWARD
 thrusterparam, 33
THGROUP_ATT_LEFT
 thrusterparam, 33
THGROUP_ATT_PITCHDOWN
 thrusterparam, 33
THGROUP_ATT_PITCHUP
 thrusterparam, 33
THGROUP_ATT_RIGHT
 thrusterparam, 33
THGROUP_ATT_UP
 thrusterparam, 33
THGROUP_ATT_YAWLEFT
 thrusterparam, 33
THGROUP_ATT_YAWRIGHT
 thrusterparam, 33
THGROUP_HANDLE
 handle, 20
THGROUP_HOVER
 thrusterparam, 33
THGROUP_MAIN
 thrusterparam, 33
THGROUP_RETRO
 thrusterparam, 33
THGROUP_TYPE
 thrusterparam, 33
THGROUP_USER
 thrusterparam, 33
Thruster and thruster-group parameters, 32
THRUSTER_HANDLE
 handle, 20
ThrusterGroupDefined
 VESSEL, 423
thrusterparam
 ENGINE_ATTITUDE, 33
 ENGINE_HOVER, 33
 ENGINE_MAIN, 33
 ENGINE_RETRO, 33
 ENGINETYPE, 33
 THGROUP_ATT_BACK, 33
 THGROUP_ATT_BANKLEFT, 33
 THGROUP_ATT_BANKRIGHT, 33
 THGROUP_ATT_DOWN, 33
 THGROUP_ATT_FORWARD, 33
 THGROUP_ATT_LEFT, 33
 THGROUP_ATT_PITCHDOWN, 33
 THGROUP_ATT_PITCHUP, 33
 THGROUP_ATT_RIGHT, 33

THGROUP_ATT_UP, 33
THGROUP_ATT_YAWLEFT, 33
THGROUP_ATT_YAWRIGHT, 33
THGROUP_HOVER, 33
THGROUP_MAIN, 33
THGROUP_RETRO, 33
THGROUP_TYPE, 33
THGROUP_USER, 33
Time functions, 98
Title
 MFD, 281
 MFD2, 288
tmul
 vec, 28
ToggleAttitudeMode
 VESSEL, 428
ToggleNavmode
 VESSEL, 375
TOP
 oapi::Sketchpad, 318
Top-level module callback functions, 183

UNDERLINE
 oapi::Font, 219
Undock
 VESSEL, 460
unit
 vec, 29
UnregisterAnimation
 VESSEL, 447
UnregisterVisObject
 oapi::GraphicsClient, 230
Update
 MFD, 281
 MFD2, 288
User input functions, 165
UserInput
 oapiOpenInputBox, 165
Utility
 oapiGetColour, 164
 oapiRand, 164
Utility functions, 164

vdata
 VESSELSTATUS, 510
vec
 _M, 22
 _V, 22
 crossp, 23
 dist, 23
 dotp, 23
 length, 23
 mul, 24
 normalise, 24
operator *-, 24, 25
operator *=, 25
operator+, 25, 26
operator+=, 26
operator-, 26, 27
operator-=, 27
operator/, 27
operator/=, 28
outerp, 28
tmul, 28
unit, 29
veccpy, 29
veccpy
 vec, 29
VECTOR3, 331
Vectors and matrices, 20
Version
 CELBODY, 201
 oapi::ModuleNV, 297
 VESSEL, 356
VESSEL, 332
 ActivateNavmode, 374
 AddAnimationComponent, 448
 AddBeacon, 473
 AddExhaust, 465–467
 AddExhaustStream, 470, 471
 AddForce, 402
 AddMesh, 439, 440
 AddParticleStream, 470
 AddPointLight, 475
 AddReentryStream, 471
 AddSpotLight, 475
 AttachChild, 464
 AttachmentCount, 463
 ClearAirfoilDefinitions, 388
 ClearAttachments, 462
 ClearBeacons, 474
 ClearControlSurfaceDefinitions, 391
 ClearDockDefinitions, 457
 ClearLightEmitters, 477
 ClearMeshes, 439, 480
 ClearPropellantResources, 403
 ClearThrusterDefinitions, 410
 ClearVariableDragElements, 393
 CopyMeshFromTemplate, 445
 Create, 482
 CreateAirfoil, 384
 CreateAirfoil2, 385
 CreateAirfoil3, 386
 CreateAnimation, 447
 CreateAttachment, 461
 CreateControlSurface, 388
 CreateControlSurface2, 389
 CreateControlSurface3, 390

CreateDock, 456
CreatePropellantResource, 402
CreateThruster, 408
CreateThrusterGroup, 419
CreateVariableDragElement, 392
DeactivateNavmode, 374
DefSetState, 367
DefSetStateEx, 368
DelAirfoil, 388
DelAnimation, 448
DelAnimationComponent, 449
DelAttachment, 461
DelBeacon, 474
DelControlSurface, 390
DelDock, 456
DelExhaust, 468
DelExhaustStream, 472
DelLightEmitter, 476
DelMesh, 441
DelPropellantResource, 403
DelThruster, 409
DelThrusterGroup, 420, 478
DetachChild, 465
Dock, 459
DockCount, 458
DockingStatus, 459
EditAirfoil, 387
EnableIDS, 433
EnableTransponder, 432
GetADCtrlMode, 373
GetAirfoilParam, 386
GetAirspeed, 383
GetAltitude, 379
GetAngularAcc, 370
GetAngularMoment, 371
GetAngularVel, 370
GetAnimPtr, 450
GetAOA, 384
GetApDist, 379
GetArgPer, 378
GetAtmDensity, 381
GetAtmPressure, 382
GetAtmRef, 381
GetAtmTemperature, 381
GetAttachmentHandle, 464
GetAttachmentId, 463
GetAttachmentIndex, 464
GetAttachmentParams, 462
GetAttachmentStatus, 463
GetAttitudeLinLevel, 429
GetAttitudeMode, 427
GetAttitudeRotLevel, 428
GetBank, 380
GetBankMomentScale, 479
GetBeacon, 475
GetCameraDefaultDirection, 436
GetCameraOffset, 435
GetClassName, 357
GetClipRadius, 360
GetCOG_elev, 362
GetControlSurfaceLevel, 392
GetCrossSections, 364
GetCW, 393
GetDamageModel, 358
GetDefaultPropellantResource, 407
GetDevMesh, 444
GetDockHandle, 458
GetDockParams, 458
GetDockStatus, 458
GetDrag, 399
GetDragVector, 400
GetDynPressure, 382
GetEditorModule, 357
GetElements, 375, 376
GetEmptyMass, 362
GetEnableFocus, 358
GetEquPos, 381
GetExhaustCount, 468
GetExhaustLevel, 469
GetExhaustSpec, 468, 469
GetFlightModel, 358
GetFlightStatus, 368
GetForceVector, 401
GetFuelMass, 408
GetFuelRate, 408
GetGlobalOrientation, 371
GetGlobalPos, 369
GetGlobalVel, 369
GetGravityGradientDamping, 366
GetGravityRef, 375
GetGroupThruster, 422, 423
GetGroupThrusterCount, 422
GetHandle, 357
GetHorizonAirspeedVector, 383
GetIDS, 434
GetISP, 418
GetLift, 399
GetLiftVector, 400
GetLightEmitter, 476
GetLinearMoment, 370
GetMachNumber, 382
GetManualControlLevel, 426
GetMass, 368
GetMaxFuelMass, 407
GetMesh, 443
GetMeshCount, 443
GetMeshName, 444
GetMeshOffset, 443

GetMeshTemplate, 444
GetMeshVisibilityMode, 445
GetName, 357
GetNavChannel, 432
GetNavCount, 431
GetNavmodeState, 375
GetNavRecv, 480
GetNavRecvFreq, 432
GetNavSource, 435
GetNosewheelSteering, 472
GetPeDist, 378
GetPitch, 380
GetPitchMomentScale, 396
GetPMI, 365
GetPropellantCount, 403
GetPropellantEfficiency, 405
GetPropellantFlowrate, 406
GetPropellantHandleByIndex, 403
GetPropellantMass, 404
GetPropellantMaxMass, 404
GetRelativePos, 369
GetRelativeVel, 370
GetRotationMatrix, 453
GetRotDrag, 395
GetShipAirspeedVector, 383
GetSize, 359
GetSlipAngle, 384
GetSMi, 378
GetStatus, 367
GetStatusEx, 367
GetSuperstructureCG, 452
GetSurfaceRef, 379
GetThrusterCount, 410
GetThrusterDir, 412
GetThrusterGroupHandle, 421
GetThrusterGroupLevel, 426
GetThrusterHandleByIndex, 410
GetThrusterIsp, 415
GetThrusterIsp0, 414
GetThrusterLevel, 416
GetThrusterMax, 413, 414
GetThrusterMax0, 412
GetThrusterMoment, 418
GetThrusterRef, 411
GetThrusterResource, 410
GetThrustVector, 400
GetTorqueVector, 401
GetTotalPropellantFlowrate, 406
GetTotalPropellantMass, 405
GetTouchdownPoints, 363
GetTransponder, 434
GetTrimScale, 397
 GetUserThrusterGroupCount, 423
 GetUserThrusterGroupHandleByIndex, 421
GetWeightVector, 399
GetWheelbrakeLevel, 473
GetWingAspect, 394
GetWingEffectiveness, 395
GetYaw, 380
GetYawMomentScale, 397
Global2Local, 455
GlobalRot, 453
GroundContact, 372
HorizonInvRot, 454
HorizonRot, 454
IncEngineLevel, 478
IncThrusterGroupLevel, 424
IncThrusterGroupLevel_SingleStep, 425
IncThrusterLevel, 417
IncThrusterLevel_SingleStep, 418
InitNavRadios, 431
InsertMesh, 440, 441
LightEmitterCount, 476
Local2Global, 454
Local2Rel, 455
MeshgroupTransform, 446
MeshModified, 446
NonsphericalGravityEnabled, 373
OrbitStabilised, 372
ParseScenarioLine, 482
ParseScenarioLineEx, 477
Playback, 451
RecordEvent, 451
Recording, 450
RegisterAnimation, 447
SaveDefaultState, 481
SendBufferedKey, 430
SetADCctrlMode, 374
SetAlbedoRGB, 361
SetAngularVel, 371
SetAnimation, 450
SetAttachmentParams, 462
SetAttitudeLinLevel, 430
SetAttitudeMode, 427
SetAttitudeRotLevel, 429
SetBankMomentScale, 479
SetCameraCatchAngle, 437
SetCameraDefaultDirection, 436
SetCameraMovement, 438
SetCameraOffset, 435
SetCameraRotationRange, 437
SetCameraShiftRange, 438
SetClipRadius, 361
SetCOG_elev, 480
SetControlSurfaceLevel, 391
SetCrossSections, 364
SetCW, 393
SetDefaultPropellantResource, 407

SetDockMode, 460
SetDockParams, 457
SetElements, 377
SetEmptyMass, 362
SetEnableFocus, 359
SetEngineLevel, 477
SetExhaustScales, 478
SetFuelMass, 408
SetGlobalOrientation, 372
SetGravityGradientDamping, 366
SetIDSChannel, 433
SetISP, 419
SetLiftCoeffFunc, 398
SetMaxFuelMass, 407
SetMaxWheelbrakeForce, 473
SetMeshVisibilityMode, 445
SetMeshVisibleInternal, 481
SetNavChannel, 431
SetNavRecv, 480
SetNosewheelSteering, 472
SetPitchMomentScale, 396
SetPMI, 365
SetPropellantEfficiency, 406
SetPropellantMass, 405
SetPropellantMaxMass, 404
SetReentryTexture, 469
SetRotationMatrix, 453
SetRotDrag, 396
SetSize, 360
SetSurfaceFrictionCoeff, 364
SetThrusterDir, 412
SetThrusterGroupLevel, 424
SetThrusterIsp, 416
SetThrusterLevel, 417
SetThrusterLevel_SingleStep, 417
SetThrusterMax0, 413
SetThrusterRef, 411
SetThrusterResource, 411
SetTouchdownPoints, 363
SetTransponderChannel, 433
SetTrimScale, 398
SetVisibilityLimit, 360
SetWheelbrakeLevel, 473
SetWingAspect, 394
SetWingEffectiveness, 395
SetYawMomentScale, 397
ShiftCentreOfMass, 451
ShiftCG, 452
ShiftMesh, 442
ShiftMeshes, 442
ThrusterGroupDefined, 423
ToggleAttitudeMode, 428
ToggleNavmode, 375
Undock, 460
UnregisterAnimation, 447
Version, 356
VESSEL, 356
Vessel creation and destruction, 60
Vessel functions, 64
Vessel mesh visibility flags, 43
Vessel module callback functions, 184
VESSEL2, 482
 clbkADCtrlMode, 491
 clbkAnimate, 493
 clbkConsumeBufferedKey, 494
 clbkConsumeDirectKey, 494
 clbkDockEvent, 493
 clbkDrawHUD, 490
 clbkFocusChanged, 487
 clbkHUDMode, 492
 clbkLoadGenericCockpit, 495
 clbkLoadPanel, 495
 clbkLoadStateEx, 486
 clbkLoadVC, 497
 clbkMFDMode, 492
 clbkNavMode, 493
 clbkPanelMouseEvent, 496
 clbkPanelRedrawEvent, 496
 clbkPlaybackEvent, 489
 clbkPostCreation, 487
 clbkPostStep, 488
 clbkPreStep, 488
 clbkRCSMode, 491
 clbkSaveState, 486
 clbkSetClassCaps, 485
 clbkSetStateEx, 487
 clbkVCMouseEvent, 498
 clbkVCRedrawEvent, 498
 clbkVisualCreated, 489
 clbkVisualDestroyed, 490
 VESSEL2, 485
VESSEL3, 499
 clbkDrawHUD, 506
 clbkGeneric, 505
 clbkGetRadiationForce, 508
 clbkLoadPanel2D, 505
 clbkPanelMouseEvent, 504
 clbkPanelRedrawEvent, 504
 clbkRenderHUD, 507
 RegisterPanelArea, 503
 RegisterPanelMFDGeometry, 502
 SetPanelBackground, 501
 SetPanelScaling, 502
 VESSEL3, 501
vessel_clbk
 ovcExit, 185
 ovcInit, 185
VesselCreation

oapiCreateVessel, 60
oapiCreateVesselEx, 60
oapiDeleteVessel, 61
VESSELSTATUS, 508
 flag, 510
 status, 510
 vdata, 510
VESSELSTATUS2, 510
 arot, 514
 flag, 513
 status, 513
 surf_hdg, 514
 surf_lat, 514
 surf_lng, 514
VESSELSTATUS2::DOCKINFOSPEC, 514
VESSELSTATUS2::FUELSPEC, 515
VESSELSTATUS2::THRUSTSPEC, 515
Virtual cockpit functions, 145
VirtualCockpit
 oapiVCRegisterArea, 145
 oapiVCRegisterHUD, 146
 oapiVCRegisterMFD, 146
 oapiVCSetAreaClickmode_Quadrilateral, 147
 oapiVCSetAreaClickmode_Spherical, 147
 oapiVCSetNeighbours, 148
 oapiVCTriggerRedrawArea, 148
VISHANDLE
 handle, 20
Visual and mesh functions, 109

WriteStatus
 MFD, 285