ORBITER File Formats

29 August 2010
Orbiter home: orbit.medphys.ucl.ac.uk/ or www.orbitersim.com

Contents

1 Introduction

This document contains information about the various file formats used by Orbiter, including configuration, scenario and mesh files. Orbiter functionality can be modified and extended by editing or adding new configuration files, e.g. to define a new spacecraft type, or to modify its visual appearance.

2 ORBITER configuration files

Configuration files allow the customisation of various aspects of Orbiter. Configuration files have file extension .cfg. They are ASCII text files which can be edited with any text editor capable of writing plain text files (e.g `notepad`).

Each line contains an item and its value, using the format

```
<item> = <value>
```

A semicolon starts a comment, continuing to the end of line.

All configuration files except for the master file (see below) are located in a subdirectory tree defined by the ConfigDir entry in the master file, usually ".\Config".

2.1 Master configuration file

The master configuration file *Orbiter.cfg* is located in the Orbiter main directory. It contains general settings for graphics modes, subdirectory locations, simulation parameters, etc. Most of the options in this file are accessible via the Orbiter Launchpad dialog box, and manual editing of the file should generally not be necessary.

Orbiter overwites the master configuration file at the start and end of each simulation session, to store any changes made in the Launchpad by the user.

By default, only entries whose values differ from their default setting are written to Orbiter.cfg. To force Orbiter to write out all values (useful for debugging or manually editing the file), open Orbiter.cfg in a text editor, change the value of *EchoAllParams* to *TRUE*, and save. Subsequently, Orbiter will write all configuration values to the file.

Item	Type	Description
EchoAllParams	Bool	If TRUE, Orbiter writes all configuration parameters to Orbiter.cfg, including defaults. Default: FALSE
LPadRect	Rect	Screen position of the launchpad dialog window (pixels)
ConfigDir	String	Subdirectory for configuration files. Default: .\Config\
MeshDir	String	Subdirectory for mesh files. Default: .\Meshes\
TextureDir	String	Subdirectory for textures. Default: .\Textures\
HightexDir	String	Subdirectory for alternative high-resolution planetary textures. Default: .\Textures2\
ScenarioDir	String	Subdirectory for scenarios. Default: .\Scenarios\
StartPaused	Bool	Suspend simulation on launch. Default: FALSE
FocusFollowsMouse	Bool	If TRUE, dialog windows receive input focus when the mouse moves over them. Default: TRUE
FlightModel	Int	Flight model realism level. Currently supported: 0 (simple) and 1 (complex). Default: 1
DamageModel	Int	Damage realism level. Currently supported: 0 (no damage) and 1 (damage modelling enabled). Default: 0
UnlimitedFuel	Bool	Ignore spacecraft fuel consumption. Default: FALSE
RefuelOnPad	Bool	Auto-refuel spacecraft parked on a landing pad. Default: TRUE
MFDTransparent	Bool	Make multifunctional displays transparent in "glass cockpit" mode. Default: FALSE
GenericMFDSize	Int	Scaling factor for MFD displays in "glass cockpit" mode. Supported values 1-10. Default: 6
MFDMapVersion	Int	Display style for Map MFD (0 = old, 1 = new). Default: 1

InstrumentUpdate-Interval	Float	Interval between MFD display updates (seconds). Default: 1
PanelScale	Float	Scaling factor for instrument panel display. Default: 1
PanelScrollSpeed	Float	Speed factor for panel scrolling (pixels per second). Default: 300
EnableShadows	Bool	Enable/disable object shadows on planet surfaces. Default: TRUE
EnableVesselSha-dows	Bool	Enable/disable vessel shadows on planet surfaces. Default: TRUE
EnableClouds	Bool	Enable rendering of planetary cloud layers. Default: TRUE
EnableCloudShad-ows	Bool	Enable rendering of cloud shadows on the ground (also requires CloudShadowDepth < 1 in individual planet con-fig files). Default: FALSE
EnableNightlights	Bool	Enable rendering of night lighting effects of planetary sur-faces. Default: TRUE
EnableWaterReflec-tion	Bool	Enable rendering of specular reflections from oceanic surfaces. Default: TRUE
EnableSpecularRip-ples	Bool	Enable microtextures on water surfaces for ripple effects. Default: FALSE
EnableHorizonHaze	Bool	Enable rendering of atmospheric effects at the horizon. Default: TRUE
EnableDistanceFog	Bool	Enable distance-dependent fog effects. Default: TRUE
EnableSpecularRef-lection	Bool	Enable specular reflection effects from polished surfaces. Default: TRUE
EnableReentry-Flames	Bool	Enable shockwave effects during reentry. Default: TRUE
EnableParticle-Streams	Bool	Enable particle generation for exhaust and reentry effects. Default: TRUE
EnableLocalLights	Bool	Enable localised point and spot light emitters. Default: FALSE
MaxLights	Int	Max. simultaneously active lights (0=query device). Default: 0
AmbientLevel	Int	Ambient light level (brightness of not directly lit surfaces). Valid range is 0-255. Default: 10
PlanetMaxPatch-Level	Int	Max. texture resolution level for planetary surfaces. Range: 1-14. Default: 14
PlanetPatchRes	Float	Texture resolution bias for planet surfaces. Range: 0.1 to 10. Higher values produce higher resolution planetary surfaces at a given apparent radius, but reduce perform-ance. Default: 1.0
NightlightBrightness	Float	Brightness level of night lighting effects. Range: 0-1. Default: 0.5
StarPrm	List	Brightness and scaling parameters for stars rendered as pixels. Values: app. mag. limit for brightest stars / app. mag. limit for dimmest stars / render brightness for dimmest stars / lin-log scaling flag. Default: [0.0 7.0 0.1 0]
CSphereBgImage	String	File name for celestial sphere background image. Default: <none>
CSphereBGPath	String	File path to celestial sphere background image.
CSphereBGIntensity	Float	Brightness of background image. Range: 0-1. Default: 0.5
Planetarium	Int	Bit flags for display elements in “Planetarium” mode (F9). Default: 4330
BodyForces	List	Display parameters for force vector visualisation. Values: Bit flags for force types / vector scale factor / opacity. Default: [60 1.0 1.0]
CoordinateAxes	List	Display parameters for object axis visualisation. Values: Bit flags for object types / axis scale factor / opacity. Default: [4 1 1]
ShutdownMode	Int	Simulation shutdown method (0=dealloc memory, 1=respawn, 2=terminate). Default: 0

FixedStep	Float	Assigns a fixed time interval per frame [s]. Default: 0 (disable fixed frame intervals)
TimerMode	Int	Simulation timer mode (0=auto, 1=hires hardware timer, 2=lores software timer). Default: 0
DistributedVessel-Mass	Bool	Enables gravity gradient torque effects as result of anisotropic inertia tensor. Default: FALSE
NonsphericalGravity-Sources	Bool	Enables orbit perturbations due to nonspherical gravitational potentials. Default: FALSE
RadiationPressure	Bool	Enables orbit perturbations due to radiation pressure. Default: FALSE
StabiliseOrbits	Bool	Use Encke's method for improved state propagation stability at large time steps. Default: TRUE
StabilisePLimit	Float	Field perturbation limit for orbit stabilisation. Default: 0.05
StabiliseSLimit	Float	Fractional orbit step limit for orbit stabilisation. Default: 0.01
PertProp-Subsampling	List	Orbit stabilisation subsampling parameters. Values: max. steps / fractional orbit step limit. Default: [10 0.02]
PertPropNon-sphericalLimit	Float	Fractional orbit step beyond which nonspherical gravity effects are ignored. Default: 0.05
LinPropStages	Int	Number of integrator stages for linear vessel propagation. Range: 1-5. Default: 4
LinPropStage<i>	List	Integrator parameters for linear propagator stage <i> (0-4). Values: Integrator index / time step limit. Default:i=0: [2 0.5], i=1: [4 20.0], i=2: [6 100.0], i=3: [8 N/A]
AngPropStages	Int	Number of integrator stages for angular vessel propagation. Range: 1-5. Default: 4
AngPropStage<i>	List	Integrator parameters for angular propagator stage <i> (0-4). Values: Integrator index / time step limit / angular step limit. Default: i=0: [2 1.0 0.017], i=1: [4 10.0 0.070], i=2: [6 100.0 0.175], i=3: [8 N/A N/A]
AngPropSub-sampling	List	Angular subsampling parameters. Values: max. steps / angular step limit. Default: [100 0.349]
AngPropLimits	List	Angular step limits for suppression of model features. Values: limit of angular moment cross-term suppression / limit of external torque suppression. Default: [0.524 62.83]
PlanetPreloadMode	Int	Planetary texture load mode. 0=load on demand, 1=preload at simulation start. Default: 0
PlanetTexLoadFreq	Float	Texture patch loading frequency [Hz]. Default: 20
PlanetAnisoMode	Int	Planet anisotropic filter level (1=none). Default: 1
PlanetMipmapMode	Int	Planet texture mipmap mode (0=none, 1=point sampling, 2=linear interpolation). Default: 1
PlanetMipmapBias	Float	Mipmap level bias. Range: -1 to 1, where < 0 is sharper, > 0 is smoother. Default: 0
CameraPanspeed	Float	Camera speed in ground observer mode. Default: 100
HUDColIdx	Int	HUD colour index. Default: 0 (green)
DeviceIndex	Int	Enumeration index for current 3D device *(do not edit manually)*
ModeIndex	Int	Screen mode index *(do not edit manually)*
DeviceForceEnum	Bool	If TRUE, enumerate 3D devices at each start. Default: TRUE
Fullscreen	Bool	TRUE for fullscreen mode, FALSE for windowed mode. Default: FALSE
Stereo	Bool	*Currently not used.*
NoVSync	Bool	Disable vertical refresh synchronisation. Default: FALSE
StencilBuffer	Bool	Use stencil buffering for semi-opaque shadows, if supported. Default: FALSE
FullscreenPageflip	Bool	Enable hardware page-flipping in fullscreen mode. Default: TRUE
WindowWidth	Int	Horizontal window size for windowed modes [pixel].
WindowHeight	Int	Vertical window size for windowed modes [pixel].

JoystickIndex	Int	Enumeration index for current joystick (0=none). Default: 0
JoystickThrottleAxis	Int	Axis index for joystick throttle. (0=Z, 1=slider 0, 2=slider 1). Default: 1
JoystickThrottleSaturation	Int	Saturation zone for joystick throttle control (0–10000). A setting of 9000 means that the throttle will saturate over the last 10% of its range at either end. Default: 9500
JoystickDeadzone	Int	Deadzone at joystick axis centres (0-10000). A setting of 2000 means the joystick is considered neutral within 20% from the central position. Default: 2500
IgnoreThrottleOnStart	Bool	Ignore throttle at simulation start until moved. Default: TRUE
DemoMode	Bool	Start Orbiter in demo mode (auto-launch scenarios). Default: FALSE
BackgroundImage	Bool	Cover screen background with an image in demo mode. Default: FALSE
BlockExit	Bool	Don't allow users to exit Orbiter in demo mode (default: FALSE)
MaxDemoTime	Float	Max. simulation runtime in demo mode (seconds). Default: 300
MaxLaunchpadIdleTime	Float	Max. time for launchpad to be open before auto-launching a scenario (seconds). Default: 15
RecordPosFrame	Int	Flight recorder: reference frame for position data (0=ecliptic, 1=equatorial). Default: 1
RecordAttFrame	Int	Flight recorder: reference frame for attitude data (0=ecliptic, 1=equatorial). Default: 1
RecordTimeWarp	Bool	Save time acceleration events in recording stream. Default: TRUE
RecordFocusEvent	Bool	Save vessel focus changes in recording stream. Default: TRUE
ReplayTimeWarp	Bool	Set time acceleration during playback from stream data. Default: TRUE
ReplayFocusEvent	Bool	Set vessel focus during playback from stream data. Default: TRUE
ReplayCameraEvent	Bool	Set camera parameters during playback from stream data. Default: TRUE
SystimeSampling	Bool	Use system time (rather than simulation time) for recording sample intervals. Default: TRUE
PlaybackNotes	Bool	Display onscreen annotations from stream data during playback. Default: TRUE
DialogFont_Scale	Float	Scaling factor for dialog font size. Default: 1.0
DialogFont1_Face	String	Standard dialog font face. Default: Arial
ActiveModules	List	List of active plugin modules

2.2 Planetary systems

Planetary systems contain stars, planets and moons. Each planetary system requires at least one star. Stars, planets and moons are defined in the planetary system's configuration file.

General parameters

Item	Type	Description
Name	String	A name for the planetary system
MarkerPath	String	Directory path containing surface marker lists for the planet. Default: `.\Config\`<name>`\Marker\`

See also Section 2.4 on how to add celestial markers to a planetary system.

Object list

The object list defines the celestial bodies populating the planetary system, and their hierarchy.

Star entries:

```
Star<i> = <Name>
```

where <*i*> is an index running from 1 upward. (note: planetary systems with more than one central star are not currently supported).

Planet entries:

```
Planet<i> = <Name>
```

where <*i*> is an index running from 1 upward.

Moon entries:

```
<Planet>:Moon<i> = <Name>
```

where <*Planet*> is the name of a planet defined before, and <*i*> is an index enumerating the moons of this planet, running from 1 upward.

Example:

```
Star1 = Sun
Planet1 = Mercury
Planet2 = Venus
Planet3 = Earth
Earth:Moon1 = Moon
Planet4 = Mars
Mars:Moon1 = Phobos
Mars:Moon2 = Deimos
```

2.3 Planets

Planet configuration files define the planet's orbital, physical and visual parameters. For an example see Config\Earth.cfg.

General parameters

Item	Type	Description
Name	String	Planet name
Module	String	Name of dynamic link library performing calculations for the planet (default: none)
ErrorLimit	Float	Max. rel. error for position/velocity calculations (only used if the module supports precision adjustment)
EllipticOrbit	Bool	If TRUE, use analytic 2-body solution for planet position/velocity calculation, otherwise update dynamically (ignored if module supports position/velocity calculation)
HasElements	Bool	If TRUE, the initial position/velocity is calculated from the provided set of orbital elements, otherwise from an explicit position/velocity pair (ignored if module supports position/velocity calculation)

Notes:

- If the module calculates the planet position and velocity from perturbation terms, then the value of ErrorLimit will affect the number of terms used for the calculation. A lower value will increase the number of required terms, and thus the calculation time. The valid range for ErrorLimit depends on the module, but is typically 1e-3 ≤ ErrorLimit ≤ 1e-8.

Orbital parameters (Ignored if module supports position/velocity calculation or HasElements = FALSE)

Item	Type	Description
Epoch	Float	Orbital element reference epoch (e.g. 2000)
ElReference	Flag	`ParentEquator` or `Ecliptic`: orbit reference frame (default: Ecliptic)
SemiMajorAxis	Float	Orbit semi-major axis [m]
Eccentricity	Float	Orbit eccentricity
Inclination	Float	Orbit inclination against reference plane [rad]
LongAscNode	Float	Longitude of ascending node [rad]
LongPerihelion	Float	Longitude of periapsis [rad]
MeanLongitude	Float	Mean longitude at epoch [rad]

Physical parameters

Item	Type	Description
Mass	Float	Planet mass [kg]
Size	Float	Mean planet radius [m]

NEW **Rotation and precession elements** (see also Doc/Technotes/precession.pdf)

Item	Type	Description
SidRotPeriod	Float	Siderial rotation period [s] (default: infinite)
SidRotOffset	Float	Rotation at epoch [rad] (default: 0)
Obliquity	Float	Obliquity of axis: angle between planet axis and precession reference axis [rad] (default: 0)
LAN	Float	Longitude of ascending node of equatorial plane [rad] (default: 0)
LAN_MJD	Float	Reference date for LAN [MJD] (default: 51544.5)
PrecessionPeriod	Float	Period of precession of axis [days] (default: infinite, i.e. no precession)
PrecessionObliquity	Float	Obliquity of precession reference axis with respect to ecliptic normal (J2000) [rad] (default: 0)
PrecessionLAN	Float	Longitude of ascending node of precession reference plane [rad] (default: 0)

Atmospheric parameters (only required if planet has atmosphere)

Item	Type	Description
AtmPressure0	Float	(Mean) atmospheric pressure at zero altitude [Pa]
AtmDensity0	Float	(Mean) atmospheric density at zero altitude [kg/m^3]
AtmGasConstant	Float	specific gas constant [J K^{-1} kg^{-1}]. Default: 286.91 (Earth value)
AtmGamma	Float	ratio of specific heats c_p/c_v. Default: 1.4 (Earth value)
AtmColor0	Vec$_3$	RGB triplet for atmospheric colour at ground level (0-1 each)
AtmAltLimit	Float	altitude limit beyond which atmospheric effects can be ignored [m]
AtmHazeExtent	Float	Width parameter for extent of horizon haze rendering. Range: 0 (thinnest) to 1 (widest). Default: 0.1
AtmHazeShift	Float	Shifts the reference altitude of the haze base line. Can be used to adjust haze altitude to a cloud layer. (in units of planet radius). Default: 0 (align with surface horizon). Shift is not applied if camera is below cloud layer.

AtmHazeDensity	Float	Modifies the density at which the horizon haze is rendered (basic density is calculated from atmospheric density) Default: 1.0
AtmHazeColor	Vec_3	RGB triplet for horizon haze colour (0-1 each). Default: use AtmColor0 values.
AtmHorizonAlt	Float	altitude scale for horizon haze rendering [m]. Default: 0.01 of planet radius.
ShadowDepth	Float	Depth ("blackness") of object shadows (0 ... 1, where 0=black, 1=no shadows). Default: exp($-\rho_0/2$), where ρ_0 is the atmospheric density at the surface. This option is only used when stencil buffering is enabled. Otherwise shadows are always black.

Cloud parameters (only required if planet contains a cloud layer)

Item	Type	Description
CloudAlt	Float	Altitude of cloud layer [m]
CloudShadowDepth	Float	Depth ("blackness") of cloud shadows on the ground (0 ... 1) where 0 = black, 1 = don't render shadows. Default: 1
CloudRotPeriod	Float	Rotation period of cloud layer against surface [s] (default: 0 – static cloud layer)
CloudMicrotextureAlt	Float+ Float	Altitude range [m] for cloud microtexturing. First value is altitude at which full microtexture is applied. Second value is altitude at which microtexture starts to kick in. First value ≥ 0 and second value > first value is required. Default: no microtexture.

Visualisation parameters

Item	Type	Description
MaxPatchResolution	Int	Max. resolution level for surface texture maps (1 ... 10)
MinCloudResolution	Int	Min. resolution at which clouds are rendered as separate layer (1 ... 8)
MaxCloudResolution	Int	Max. cloud resolution level (MinCloudResolution ... 8)
SpecularRipple	Bool	If TRUE, and if "Specular ripples" option is enabled in the Launchpad dialog, specularly reflecting surfaces use a "water ripple" microtexture. Default: FALSE.

Surface marker parameters (optional)

Item	Type	Description
Marker-Path	String	Directory path containing surface marker lists for the planet. Default: `.\Config\`<planet name>`\Marker\`

See also Section 2.4 on how to add surface markers to a planet.

Surface bases (optional)

This list contains the names and locations of surface landing sites ("spaceports"). Each entry in this list must be accompanied by a configuration file for the corresponding surface base.

```
BEGIN_SURFBASE
        <base list>
END_SURFBASE
```

Base list entries have the following format:

```
<name>: <lng> <lat>
```

where

<name>	Name which identifies the base config file (*<name>*.cfg). The actual name of the base as it appears in Orbiter is given by the NAME tag in the base config file.
<lng> <lat>	Base position (equatorial coordinates) [deg]

Note that there is an alternative format for this list, using a NumBases entry and BaseXX tags. This format is obsolete and should no longer be used.

Ground-based observer sites (optional)

This list contains the pre-defined locations for ground-based observers (launch cameras, spectators, etc.) which can be selected in the Camera dialog. The format of the list is

```
BEGIN_OBSERVER
	<observer list>
END_OBSERVER
```

List entries have the following format:

```
<site>:<spot>:  <lng> <lat> <alt>
```

where

<site>	a name which identifies the site (e.g. KSC)
<spot>	the particular location at the site (e.g. Launch pad 39)
<lng> <lat>	observer position (equatorial coordinates) [deg]
<alt>	observer altitude [m] (>0)

The easiest way to find the coordinates for a new observer spot is to open the *Camera* dialog (Ctrl F1), and select a nearby location under the *Ground* tab. Then move the camera to the new spot using Ctrl ↓ ↑ → ← and Page Up Page Down. The coordinates are displayed in the dialog and can be directly copied into the configuration file.

Navbeacon transmitter list (optional)

This list contains all navigation radio transmitter specs except those directly associated with a spaceport (see section 0). The list format is as follows:

```
BEGIN_NAVBEACON
	<NAV list>
END_NAVBEACON
```

List entries have the following format:

```
<type> <id> <lng> <lat> <freq> [<range>]
```

where

<type>	transmitter type. currently supported: VOR
<id>	identifer code (up to 4 letters)
<lng> <lat>	transmitter position (equatorial coordinates) [deg]
<freq>	transmitter frequency [MHz]
<range>	transmitter range [m] (default: 500 km)

To implement a custom DLL module for planet position/velocity calculations, see SDK documentation.

To add a new planet to a planetary system the following steps are required:

1. Add an entry for the planet in the planetary system configuration file (see previous section):

   ```
   Planet<X> = <Planetname>
   ```

2. Create a configuration file <*Planetname*>*.cfg* for the new planet in the “Config” subdirectory, with entries as listed above.
3. Create the required surface texture maps up to the specified resolution.
4. Optionally, create a monochrome (green on black) surface outline bitmap (256x128, BMP) to be used by the Map MFD. The file name should be <*Planetname*>*M.bmp*.

Surface bases

Surface bases (or “spaceports”) are launch and landing sites on the surface of planets or moons, usually equipped with launchpads or runways, for vertical and horizontal liftoff and/or landing. Each surface base is defined via its own configuration file. When Orbiter loads a planet configuration, it scans all surface base definitions for the planet and creates the corresponding bases.

The base definition file

To create a new surface base, you need to write a definition file for it. The file name should be of the form <*base-name*>.cfg, and it must be placed in an appropriate folder. (see *Linking surface bases to planets* below).

The format of the surface base definition file is as follows:

```
BASE-V2.0
NAME = <Base name>
LOCATION = <lng> <lat>
SIZE = <size>
OBJECTSIZE = <osize>
MAPOBJECTSTOSPHERE = [TRUE|FALSE]

BEGIN_NAVBEACON
	<NAV list>
END_NAVBEACON

BEGIN_OBJECTLIST
	<Object list>
END_OBJECTLIST

BEGIN_SURFTILELIST
	<Surface tile list>
END_SURFTILELIST
```

`BASE-V2.0`

Format identifer that must be placed in the first line of the file. This item is optional if the base is referenced directly in the planet’s base list.

`NAME = ` <*Base name*>

Defines the base’s (logical) name which need not correspond to the file name.

`LOCATION = ` <*lng*> <*lat*>

Defines the position of the base on the planet surface, where <*lng*> is longitude (deg, West < 0, East > 0), and <*lat*> is latitude (deg, South < 0, North > 0). This item is optional if the base is referenced directly in the planet's base list.

`Size = `<*size*>

Defines the base's overall radius in meters.

`OBJECTSIZE = `<*osize*>

Defines the size of a "typical" object (building, etc.) This value is used by Orbiter to determine up to what camera distance base objects will be rendered. Objects will not be rendered if the apparent size of an object of size <*osize*>, located at the centre of the base, would be smaller than 1 pixel. The default value for <*osize*> is 100.0.

`MAPOBJECTSTOSPHERE` (boolean)

If *true*, the objects in the object list will be automatically adjusted in elevation to correct for the planets curvature. This means that objects with elevation 0 will be mapped onto altitude 0 of the planet surface. If *false*, elevation 0 maps onto the flat horizon plane of the base reference point. Default: *false*.

Note: Currently this function is only implemented for a limited number of base object types.

<*NAV list*>

Contains a list navigation radio transmitters associated with the base. The format is identical to that of the Planet config file (see section 2.3).

<*Object list*>

Contains a list of objects which make up the visual elements of the base. See next section for details.

<*Surface tile list*>

An optional list of high-resolution surface tiles covering the base area. Each tile is represented by a line in the list, with the format

```
<res> <lng-idx> <lat-idx> <flag>
```

where <*res*> is the tile resolution (integer ≥ 1), and <*lng-idx*> and <*lat-idx*> are the position indices. The position indices define the location of the tile on the global planet map at the given resolution. <*flag*> is a bitflag (bit 0 = 1: render tile; bit 1 = 1: tile contains transparency in the alpha channel).

For each tile entry, a corresponding texture file in DDS format (DXT1 or DXT5) must exist in the *Textures* subdirectory, with naming convention

```
<planet>_<res>_[W|E]<lng-idx>_[N|S]<lat-idx>.dds
```

where <*planet*> is the planet name, <*res*> is the tile resolution as defined in the list, and <*lng-idx*> and <*lat-idx*> are the position indices as defined in the list (zero-padded to 4 digits).

Note: Future versions of Orbiter may incorporate local high-resolution planetary surface areas directly in the planet's texture file. The mechanism of associating surface tiles with base definitions via the surface tile list will then be removed. Using surface tile lists is therefore not recommended.

Linking surface bases to planets

After creating the base configuration file, it must be referenced by a planet to instantiate the base. There are several ways to make Orbiter read a surface base definition:

- Place the base configuration file in the default base configuration folder for the planet. By default, Orbiter will scan the folder *Config\<pname>\Base* for base definitions, where <*pname*> is the planet name. For example, the default folder for Earth bases is *Config\Earth\Base*. The default folder is only scanned if the planet doesn't explicitly define a base list (see below).
- To make Orbiter scan different folders, create a surface base list in the planet's configuration file, starting with the line *BEGIN_SURFBASE* and ending with the line *END_SURFBASE*. In this list, specify the new surface base directory with the line '*DIR* <*folder*>', where <*folder*> is the path to the folder containing the base configurations (relative to the *Config* folder). Multiple folders can be specified. If the same base is defined in more than one of the scanned folders, only the first is used. This allows to replace base definitions without having to delete the original configuration file.

  Example:

  ```
  BEGIN_SURFBASE
    DIR Earth\MyBase
    DIR Earth\MoreBases
  END_SURFBASE
  ```

  If the surface base list exists in a planet's configuration file, the default base configuration folder will *not* be scanned, unless it is explicitly listed.
- Base references can be placed directly into the base list, using the following format:

  ```
  <fname>:<lng> <lat>
  ```

  where <*fname*> is the name of the surface base configuration file: Config\<*fname*>.cfg, and <*lng*> and <*lat*> are the equatorial coordinates of the base (longitude and latitude) in degrees. When using this format, the base configuration file must be located in the *Config* subdirectory.

Selective loading of bases

To provide more control over the loading of surface bases in for a given simulation scenario, two conditional flags can be set with a DIR entry in the base list:

- Specify a time interval with the PERIOD parameter. The corresponding directory will only be scanned if the scenario start date is inside this interval. This allows to replace surface bases only at specific time periods, for example to set up the Kennedy Space Center for the Apollo lunar missions.

  Syntax:

  ```
  DIR <folder> PERIOD <mjd0> <mjd1>
  ```

  where <*mjd0*> and <*mjd1*> are the start and end dates of the period over which the base folder will be scanned, in MJD (Modified Julian Date) format. Either can be set to '–' to disable the limit at one end.
- Specify a scenario context with the CONTEXT parameters. The corresponding directory will only be scanned if the scenario specifies the same context string in its

environment context entry (see Section 3). This allows to add or replace surface bases only within a defined set of scenarios.

Syntax:

```
DIR <folder> CONTEXT <string>
```

where <*string*> is the context string to be matched against the scenario context.

The PERIOD and CONTEXT parameters can be used simultaneously. In that case the directory will only be scanned if both conditions are satisfied.

Examples:

```
BEGIN_SURFBASE
    DIR Earth\1969Base PERIOD 40222 42048
    DIR Earth\TempBases CONTEXT RichScenery
    DIR Earth\OtherBases PERIOD – 40000 CONTEXT EarlyBases
    DIR Earth\Base
END_SURFBASE
```

Note that the list order is important to allow the custom base definitions to replace standard bases.

Adding objects to surface bases

Surface bases are composed of objects (buildings, train lines, hangars, launch pads, etc.) The configuration file for each surface base contains a list of its objects:

```
BEGIN_OBJECTLIST
    <Object 0>
    <Object 1>
    …
    <Object n-1>
END_OBJECTLIST
```

Each object entry in the list defines a particular object and its properties (type, position, size, textures, etc.). An object can either be a pre-defined type or a generic mesh. Each object entry has the following format:

```
<Type>
    <Parameters>
END
```

Note that textures used by base objects must be listed in the texture list of the *Base.cfg* configuration file.

The following pre-defined object types are currently supported:

BLOCK

A 5-sided "brick" (without a floor) which can be used as a simple generic building, or as part of a more complex structure. The following parameters are supported:

Parameter	Type	Description
POS	V	Centre of the block's base rectangle (in local coordinates of the surface base). Note that the y-coordinate is the elevation above ground. Default: 0 0 0
SCALE	V	Object size in the three coordinate axes. Default: 1 1 1
ROT	F	Rotation around vertical axis (degrees). Default: 0
TEX1	S F F	Texture name and u,v scaling factors for walls along the x-axis. Default: none
TEX2	S F F	Texture name and u,v scaling factors for walls along the z-axis. Default: none
TEX3	S F F	Texture name and u,v scaling factors for roof. Default: none

(V=Vector, F=Float, S=String)

HANGAR

A hangar-type building with a barrel-shaped roof. The following parameters are supported:

Parameter	Type	Description
POS	V	Centre of the object's base rectangle (in local coordinates of the surface base). Note that the y-coordinate is the elevation above ground. Default: 0 0 0
SCALE	V	Object size in the three coordinate axes. Default: 1 1 1
ROT	F	Rotation around vertical axis (degrees). Default: 0
TEX1	S F F	Texture name and u,v scaling factors for walls. Default: none
TEX2	S F F	Texture name and u,v scaling factors for front gate. Default: none
TEX3	S F F	Texture name and u,v scaling factors for roof. Default: none

(V=Vector, F=Float, S=String)

HANGAR2

A hangar-type building with a tent-shaped roof. The following parameters are supported:

Parameter	Type	Description
POS	**V**	**Centre of the object's base rectangle (in local coordinates of the surface base). Note that the y-coordinate is the elevation above ground. Default: 0 0 0**
SCALE	**V**	**Object size in the three coordinate axes. Default: 1 1 1**
ROT	**F**	**Rotation around vertical axis (degrees). Default: 0**
TEX1	**S F F**	**Texture name and u,v scaling factors for front and back walls. Default: none**
TEX2	**S F F**	**Texture name and u,v scaling factors for side walls. Default: none**
TEX3	**S F F**	**Texture name and u,v scaling factors for roof. Default: none**
ROOFH	**F**	**Roof height from base to ridge. Default: ½ building height.**

(V=Vector, F=Float, S=String)

HANGAR3

A hangar-type building with a barrel-shaped roof reaching to the ground. The following parameters are supported:

Parameter	Type	Description
POS	V	Centre of the object's base rectangle (in local coordinates of the surface base). Note that the y-coordinate is the elevation above ground. Default: 0 0 0
SCALE	V	Object size in the three coordinate axes. Default: 1 1 1
ROT	F	Rotation around vertical axis (degrees). Default: 0
TEX1	S F F	Texture name and u,v scaling factors for front and back walls. Default: none [*not supported yet!*]
TEX2	S F F	Texture name and u,v scaling factors for front gate. Default: none [*not supported yet!*]
TEX3	S F F	Texture name and u,v scaling factors for roof. Default: none

(V=Vector, F=Float, S=String)

TANK

A fuel tank-like upright cylinder with flat top. The following parameters are supported:

Parameter	Type	Description
POS	V	Centre of the object's base circle (in local coordinates of the surface base). Note that the y-coordinate is the elevation above ground. Default: 0 0 0

SCALE	V	Cylinder radii in x and z, and height in y. Default: 1 1 1
ROT	F	Rotation around vertical axis (degrees). Default: 0
NSTEP	I	Number of segments to approximate circle. Default: 12
TEX1	S F F	Texture name and u,v scaling factors for mantle. Default: none
TEX2	S F F	Texture name and u,v scaling factors for top.

(V=Vector, F=Float, I=Integer, S=String)

RUNWAY

Texturing for a runway. The texture mapping can be split into segments, to allow inclusion of markings, overruns, etc. This does not include any lighting (see RUNWAYLIGHTS).

Parameter	Type	Description
END1	V	First end point of runway (center line), including any overruns to be textured.
END2	V	Second end point of runway (center line).
WIDTH	F	Runway width [m]
ILS1	F	Localiser frequency for approach towards END1 (108.00 to 139.95). Default: No ILS support
ILS2	F	Localiser frequency for approach towards END2 (108.00 to 139.95). This can be the same frequency as ILS1. Default: No ILS support.
NRWSEG	I	Number of texture segments
RWSEGx	I F F F F F	Definition of segment x (x = 1...NRWSEG). Parameters: 1. Number of mesh sub-segments (≥1) 2. Fractional length of segment (sum of all segments must be 1) 3. texture coordinate u_0 of segment 4. texture coordinate u_1 5. texture coordinate v_0 6. texture coordinate v_1
RWTEX	S	Texture name for all segments

(V=Vector, F=Float, I=Integer, S=String)

RUNWAYLIGHTS

Complete lighting for a single runway, including optional Precision Approach Path Indicator (PAPI) and Visual Approach Slope Indicator (VASI) – see section **Error! Reference source not found.**. Runway markers are turned off during daytime, but PAPI and VASI indicators are always active.

Parameter	Type	Description
END1	V	First end point of runway (center line).
END2	V	Second end point of runway (center line).
WIDTH	F	Runway width [m]
COUNT1	I	Number of lights along the runway center line (≥2). Default: 40
PAPI	F F F	Precision Approach Path Indicator (PAPI). Default: no PAPI. Parameters: Designated approach angle [deg] Approach cone aperture [deg] Offset of PAPI location from runway endpoints. [m]
VASI	F F F	Visual Approach Slope Indicator (VASI). Default: no VASI Parameters: Designated approach angle [deg] Distance between white and red indicator lights [m] Offset of VASI (red bar) location from runway endpoints [m]

(V=Vector, F=Float, I=Integer)

BEACONARRAY

A linear array of illuminated beacons, usable e.g. for taxiway night lighting.

Parameter	Type	Description
END1	V	First end point of beacon array (in local coordinates of the surface base). Note that the y-coordinate is the elevation above ground.
END2	V	Second end point of beacon array
COUNT	I	Number of beacons in the array (≥2). Default: 10
SIZE	F	Size (radius) of each beacon light. Default: 1.0
COL	F F F	Beacon colour (RGB) Valid range: 0..1 for each value. Default: 1 1 1 (white)

(V=Vector, F=Float, I=Integer)

SOLARPLANT

A grid of ground-mounted solar panels, smart enough to align themselves with the Sun. The following parameters are supported:

Parameter	Type	Description
POS	V	Centre position of the panel grid. Default: 0 0 0
SCALE	F	Scaling factor for each panel. Default: 1
SPACING	F F	Distance between panels in x and z direction. Default: 40 40
GRID	I I	Grid dimensions in x and z direction. Default: 2 2
ROT	F	Rotation of plant around vertical axis (degrees). Default: 0
TEX	S [F F]	Texture name and u,v scaling factors for panels. Default: none

(V=Vector, F=Float, I=Integer, S=String)

TRAIN1

A monorail-type train on a straight track. The following parameters are supported:

Parameter	Type	Description
END1	V	First end point of track
END2	V	Second end point of track
MAXSPEED	F	Maximum speed of train [m/s] Default: 30
SLOWZONE	F	Distance over which train slows down at end of track [m] Default: 100
TEX	S	Texture name

(V=Vector, F=Float, S=String)

TRAIN2

Suspension train on a straight track. The following parameters are supported:

Parameter	Type	Description
END1	V	First end point of track
END2	V	Second end point of track
HEIGHT	F	Height of suspension track over ground [m] Default: 11
MAXSPEED	F	Maximum speed of train [m/s] Default: 30
SLOWZONE	F	Distance over which train slows down at end of track [m] Default: 100
TEX	S	Texture name

(V=Vector, F=Float, S=String)

LPAD1

An octagonal bordered landing pad. Default diameter 80m (at scale 1). Landing pads are numbered in the order they appear in the list. Can be assigned numbers 1-9. For expected layout of texture map see e.g. Textures\Lpad01.dds.

Parameter	Type	Description
POS	V	Pad centre coordinates (in local coordinates of the surface base).
SCALE	F	Scaling factor. Default: 1
ROT	F	Rotation around vertical axis (degrees). Default: 0

TEX	S	Texture name. Default: none
NAV	F	frequency [MHz] of VTOL nav transmitter (valid range: 85.0-140.0, default: none)

(V=Vector, F=Float, S=String)

LPAD2

A square landing pad. Default size 80m (at scale 1). Landing pads are numbered in the order they appear in the list. Can be assigned numbers 1-99. For expected layout of texture map see e.g. Textures\Lpad02.dds.

Parameter	Type	Description
POS	V	Pad centre coordinates (in local coordinates of the surface base).
SCALE	F	Scaling factor. Default: 1
ROT	F	Rotation around vertical axis (degrees). Default: 0
TEX	S	Texture name. Default: none
NAV	F	frequency [MHz] of VTOL nav transmitter (valid range: 85.0-140.0, default: none)

(V=Vector, F=Float, S=String)

LPAD2A

Similar to LPAD2, but uses a different layout for the texture map, providing a higher resolution at the same texture size. For expected layout of texture map see e.g. Textures\Lpad02a.dds. The supported parameters are the same as for LPAD2.

MESH

Generic mesh for custom object types. Mesh files must be in ORBITER mesh file format (see *3DModel.pdf* in the Orbiter SDK package).

Parameter	Type	Description
FILE	S	Mesh file name (without path and extension). Mesh files must be located in the mesh subdirectory (see master config file).
POS	V	Position of mesh origin (in local coordinates of the surface base).
SCALE	V	Scaling factors in x and z, and height in y. Default: 1 1 1
ROT	F	Rotation around vertical axis (degrees). Default: 0
TEX	S	Texture name. Default: none
SHADOW		Render the shadow cast on the ground by the object.
UNDERSHADOWS		Object can be covered by shadows cast on the ground by other objects (e.g. roads, landing pads, etc.). Default: object not covered by ground shadows
OWNMATERIAL		Use materials and textures defined in the mesh file. This overrides the TEX entry.
LPAD		Object is a landing pad.
PRELOAD		Mesh should be loaded at program start. This can reduce disk activity during the simulation but increases main memory usage. Default: Load only when used.

(V=Vector, F=Float, S=String)

Notes:

- If the mesh only uses a single texture it is more efficient to specify it via the TEX entry than via the mesh using OWNMATERIAL, because Orbiter can merge objects with the same TEX entries for improved performance.

2.4 Adding custom markers

You can define lists of labels to mark objects on the celestial sphere (e.g. bright stars, navigation stars, nebulae, etc.), or planetary surface markers to locate natural landmarks, points of interest, historic landing sites, navigational aids, etc.

The user can display these markers during the simulation using F9 and Ctrl F9.

All celestial and planetary surface markers are placed in their own subdirectories, which default to `.\Config\`*<name>*`\Marker\`, where <*name*> is the name of the planetary system (for celestial markers) or planet (for surface markers) they are referring to. You can specify a different location with the *MarkerPath* option in the planet's or planetary system's configuration file (see Section 2.3). Marker files must have extension `.mkr`. Multiple files can be defined for a single planet or planetary system, which the user can turn on or off individually. Marker files are in ASCII (text) format:

```
BEGIN_HEADER
     InitialState [on/off]
     ShapeIdx [0 .. 6]
     ColourIdx [0 .. 5]
     Size [0.1 .. 2]
     DistanceFactor [1e-5 .. 1e3]
     Frame [celestial/ecliptic]
END_HEADER
BEGIN_DATA
     <lng> <lat> : <label> [: <label>]
     <lng> <lat> : <label> [: <label>]
     ...
```

The header section contains some configuration options:

- InitialState defines if the labels are initially visible when the user activates surface markers under Ctrl F9. The user can turn lists on and off individually during the simulation. The default is "off".
- ShapeIdx: an integer between 0 and 6 defining the shape of the labels.

0	box	(default)
1	circle	
2	diamond	
3	delta	
4	nabla	
5	cross	
6	X	

- ColourIdx: an integer between 0 and 5 defining the colour of the labels. Default is 1.
- Size: A size factor for the markers. Default is 1.0.
- DistanceFactor: Defines up to what distance the markers are displayed. Default is 1.0.
- Frame (used for celestial markers only): defines the reference frame to which the coordinates in the list refer.

ecliptic:	data are ecliptic longitude and latitude

celestial:	data are right ascension and declination of J2000 equator and equinox. (default)

Each item in the header section is optional. If missing, the default value is substituted. The header can also be omitted altogether, in which case the "BEGIN_DATA" flag is also not required.

In the data section, each line defines a label. It consists of equatorial position: longitude (in degrees, with eastern longitudes positive, and western longitudes negative), latitude (in degrees with northern latitudes positive, and southern latitudes negative), and one or two label strings to be displayed above and below the marker.

2.5 Vessel configuration files

All vessel configuration files are by default located in ORBITER's `Config` subdirectory (unless the *ConfigDir* entry in `Orbiter.cfg` points to a different directory).

Below is a description of the default vessel configuration options recognised by Orbiter. Note that not all options need to be present in a configuration file. In particular vessels defined via costomised modules may specify various parameters directly in the module. Furthermore, vessel modules may read additional custom parameters not listed here from the configuration file.

Item	Type	Description
BaseClass	S	Optional; parent class. Missing entries are taken from this class. Allows the construction of class hierarchies. (Make sure not to introduce circular dependencies!)
Module	S	Optional; name of plugin module for vessel customisation. The module must be located in the *Modules* folder.
Help	S,S	Optional; name of help file to be used for vessel class specific help when the user presses the "Vessel" button on the Help dialog. The help file must be a compiled html file (.CHM) and be located in directory Html/Vessels. The entry contains the file name without path and extension, and (separated by comma) the name of the first page of the file to be displayed (without extension). Default: no vessel class specific help.
EditorCreate	B	If *false*, the vessel type does not appear in the list on the vessel creation page of the scenario editor. (default: *true*)
ImageBmp	S	File name of a bitmap file (BMP) displaying the vessel. The name should include the path (relative to orbiter main directory) and extension (.bmp). This image is shown on the vessel creation page of the Scenario Editor. For best results, it should be size 164x240 pixels.
MeshName	S	Name of the mesh used for visualisation
EnableFocus	B	*true* if vessel can receive input focus (default: *true*)
EnableXPDR	B	*true* if vessel carries a transponder (default: *false*)
XPDR	I	transponder channel (in units of 0.05 kHz from 108.0 kHz). Only used if EnableXPDR=true. This default channel may be overridden by a vessel's scenario script.

Mass	F	Vessel mass (empty) [kg]
Size	F	(Mean) vessel radius [m]
MaxMainThrust	F	Main thruster rating [N]
MaxRetroTrust	F	Retro thruster rating [N]
MaxHoverThrust	F	Hover thruster rating [N]
MaxAttitudeThrust	F	Thrust rating for reaction contol engines [N]
TouchdownPoints	V V V	3 surface contact points in local vessel coordinates. For aircraft-like configurations these are: nose wheel, left main wheel, right main wheel. (the order is important to define the “up” direction). Other spacecraft types may interpret the points differently.
CameraOffset	V	Camera position inside the vessel for cockpit view
CW	F F F F	Airflow resistance coefficients: forward, backward, transversal, vertical. Only used by legacy flight model (if no airfoils are defined in the module).
WingAspect	F	The wing aspect ratio (wingspan^2 / wing area). Used for atmospheric drag calculation in the legacy flight model.
WingEffectiveness	F	A wing form factor: ~3.1 for elliptic wings, ~2.8 for tapered wings, ~2.5 for rectangular wings. Only used by legacy flight model.
CrossSections	V	Cross sections in axis directions (z=longitudinal) [m^2]
RotResistance	V	Resistance against rotation around axes in atmosphere, where angular deceleration due to atmospheric friction is $a^{(□)}_{x,y,z}$ = $-v^{(□)}_{x,y,z}$ □ $r_{x,y,z}$ with angular velocity $v^{(□)}$ and atmospheric density □.
Inertia	V	Principal moments of inertia, mass-normalised (see below) [m^2]
GravityGradientDamping	F	Damping coefficient for gravity gradient torque. Determines relaxation time for tidal locking. Default: 0 (undamped).
PropellantResource*i*	F [F]	Specs for propellant resource *i* ($i \geq 1$). First value: max. fuel capacity [kg]. Second value: fuel efficiency factor (>0, default: 1)
MaxFuel	F	Max. fuel mass [kg]. Obsolete; only used if no propellant resources are defined
Isp	F	Default value for fuel-specific impulse [m/s]: Amount of thrust [N] obtained by burning 1kg of fuel per second. Vessel modules can override this value for individual engines.
MEngineRef*i*	V	Reference position for main thruster *i* (*i*=1…)
REngineRef*i*	V	Reference position for retro thruster *i* (*i*=1…)
HEngineRef*i*	V	Reference position for hover thruster *i* (*i*=1…)
AttRef*dij*	V	Reference position for attitude thruster (for rotation around axis *d* (*d*=X,Y,Z), rotation direction *i* (*i*=1,2) and thruster index *j* (*j*=1,2)) for a total of 12 attitude thrusters
LongAttRef*ij*	V	Reference position for attitude thrusters (for linear forward/backward translation), direction *i* (*i*=1,2) and thruster index *j* (*j*=1,2)) for a total of 4 attitude thrusters
DockRef	V	Docking reference point for first docking port (obsolete)

DockDir	V	Docking approach direction for first docking port (obsolete)
DockRot	V	Longitudinal alignment direction (normal to DockDir) for first docking port (obsolete)
<Docklist>	List	List of positions and approach directions for docking ports (see below).
<Attachment list>	List	List of positions and approach directions for attachment points (see below).

(S=String, B=Bool, F=Float, V=Vector)

Notes:

- A vessel class can be derived from a different vessel class, by defining the BaseClass entry. All properties not defined in the new class configuration file are taken from the base class.
- The mesh name should not contain the file extension (.msh) and should not contain a directory path.
- The MaxFuel entry has been replaced by PropellantResource, which allows the definition of multiple propellant resources (fuel tanks).
- The DockRef, DockDir, DockRot entries have been replaced with the more versatile Docklist (see below), which allows the configuration of multiple docking ports and IDS frequencies.

```
BEGIN_DOCKLIST
  <Dock-spec 0>
  <Dock-spec 1>
  . . .
  <Dock-spec n-1>
END_DOCKLIST
```

where *<Dock-spec i>*:

```
<xi> <yi> <zi> <dxi> <dyi> <dzi> <rxi> <ryi> <rzi> [<ids-channel>]
```

<x_i> <y_i> <z_i> is the reference position of the docking port in the vessel's local coordinates. *<dx_i> <dy_i> <dz_i>* is the direction in which a ship approaches the docking port in the station's local reference frame.

<rx_i> <ry_i> <rz_i> is a reference direction perpendicular to the approach direction used for aligning an approaching ship's rotation along its longitudinal axis.

<ids-channel> is an optional parameter which allows to define the channel for an IDS (Instrument Docking System) transmitter for the dock. The value is an integer from which the frequency is calculated by $f = f_{\min}$ + *<ids-channel>* * 0.05 kHz, where $f_{\min}$ = 108.0 kHz.

The IDS setting can be overridden by individual vessels via the IDS option in the scenario file. Defining the IDS in the config file is usually only useful for objects with a single instance, for example space stations.

- The attachment list is similar to the docklist: it allows to specify points at which vessels can be connected to each other. Unlike docking ports, attachment points define parent-child hierarchies, and each attachment point is either a parent or a child port. For more details see the Vessel attachment management section in the API Reference Manual.

```
BEGIN_ATTACHMENT
        <Attach-spec 0>
        <Attach-spec 1>
        . . .
        <Attach-spec n-1>
END_ATTACHMENT
```

where <*Attach-spec i*>:

```
<type> <x_i> <y_i> <z_i> <dx_i> <dy_i> <dz_i> <rx_i> <ry_i> <rz_i> <id>
```

<*type*> is a single character: 'P' – "attach to a parent", or 'C' – "attach to a child".

The next 9 entries define the attachment position and direction in the same way as docking ports.

<*id*> is a string of up to 8 characters used for defining compatibility between attachment points.

- **Inertia tensor *J*:** Relates angular momentum and angular velocity: $\mathbf{L} = J \cdot \boldsymbol{\omega}$

$$J = \frac{1}{M}\int_{Vol} m(r)\begin{pmatrix} y(r)^2 + z(r)^2 & x(r)y(r) & x(r)z(r) \\ y(r)x(r) & x(r)^2 + z(r)^2 & y(r)z(r) \\ z(r)x(r) & z(r)y(r) & x(r)^2 + y(r)^2 \end{pmatrix} dr$$

where *M* is the total vessel mass, and the integration is over the vessel volume. Note that this definition normalises by *M*, so the unit of *J* is [m²]. The principal moments of inertia (PMI) J_x, J_y, J_z required by the configuration file are the diagonal elements of *J* in a reference frame in which *J* is diagonal:

$$\hat{J} = \begin{pmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{pmatrix}$$

The SDK contains a simple tool to calculate the inertia tensor for a given mesh: `Orbitersdk\utils\shipedit.exe`. The tool requires "well behaved" meshes (composed of closed compact surfaces) and assumes a homogeneous density distribution inside the mesh. The latter is not very realistic, so the results must be interpreted carefully. They should still serve as a good starting point for experimentation.

3 Scenario files

Scenarios (simulation startup definitions) contain all parameters required to set up the simulation at a particular time. They are used for loading and saving simulation states. Scenario files are usually generated automatically when saving a simulation. The format description below is primarily intended for developers of scenario editor add-ons.

Scenarios are located in a subdirectory defined by the ScenarioDir entry in the master file, usually ".\Scenarios". They have file extension .scn.

Format:

```
<Description block>
<Environment block>
<Focus block>
<Camera block>
<Panel block>
<VC block>
<HUD block>
<Left MFD block>
<Right MFD block>
<Ship list>
```

Description block (optional):

Contains a short description of the scenario.

```
BEGIN_DESC
        <Description>
END_DESC
```

<Description>: ASCII text describing the scenario. CR is converted to space. Empty lines are converted to CR. This text is displayed in the description box of the Orbiter launchpad dialog when the user selects the scenario from the list.

Environment block (optional):

Contains the simulation environment.

```
BEGIN_ENVIRONMENT
        <Environment parameters>
END_ENVIRONMENT
```

<Environment parameters>:

Parameter	Type	Description
SYSTEM	S	Name of the planetary system. A configuration file for this system must exist. Default: "Sol"
DATE		Contains simulation start time. Allowed formats are: `MJD` *<mjd>* (*<mjd>*: Modified Julian Date) `JD` *<jd>* (*<jd>*: Julian Date) `JE` *<je>* (*<je>*: Julian Epoch) Default is current system time, but this should be avoided if the scenario contains objects defined by position/velocity vectors, which cannot easily be propagated in time.

HELP	S,S	Scenario help file (in compressed HTML (CHM) format. This can be used to provide the user with additional information for the scenario. The first string contains the file name (without extension), the second contains the page (without extension) in the chm file to be opened at startup. Help files should be located in the Html\Scenarios subfolder. Scenario help files can be opened during the simulation with Alt F1. Default: no help file.
CONTEXT	S	Optional context string. This can be used to fine-tune the setup of the planetary system, e.g. by selective loading of surface bases.
NEW SCRIPT	S	A script file to run at the launch of the scenario. The string should contain any path relative to the “Script” subdirectory, but no file extension (.lua) is assumed). Default: no script.

Note: If the DATE entry is not present, Orbiter reads the computer’s system clock, adds the time zone offset to convert to Universal Time (UTC), and adds another offset of 66.184 seconds to map from UTC to Barycentric Dynamical Time (TDB).

Focus block (mandatory):

Contains parameters for the user-controlled spacecraft.

```
BEGIN_FOCUS
        <Focus parameters>
END_FOCUS
```

<Focus parameters>:

Parameter	Type	Description
SHIP	S	Name of the user-controlled ship. The ship must be listed in the ship list (see below).

Camera block (optional):

Camera mode and parameters. If the camera block is missing, the camera is set to cockpit view in the current focus object.

```
BEGIN_CAMERA
        <Camera parameters>
END_CAMERA
```

<Camera parameters>:

Parameter	Type	Description
MODE	Flag	`Extern` or `Cockpit`
TARGET	S	Camera view target. (external modes only; cockpit mode always refers to current focus object)
POS	V	Camera position relative to target (external modes only)
TRACKMODE	Flag [+String]	`TargetRelative` | `AbsoluteDirection` | `GlobalFrame` | `TargetTo` *<ref>* | `TargetFrom` *<ref>* | `Ground` *<ref>* (external modes only)
GROUNDLOCATION	F F F	longitude (deg), latitude (deg) and altitude (m) of ground observer (*Ground* trackmode only)
GROUNDDIRECTION	F F	polar coordinates of ground observer orientation (*free Ground* trackmode only)
FOV	F	Field of view (degrees)

Panel block (optional):

2D instrument panel parameters. If neither this nor the VC (virtual cockpit) block is present, Orbiter initially displays generic cockpit views.

```
BEGIN_PANEL
        <Panel parameters>
END_PANEL
```

Currently no panel parameters are supported.

VC block (optional):

Virtual cockpit parameters. If neither this nor the panel block is present, Orbiter initially displays generic cockpit views.

```
BEGIN_VC
        <VC parameters>
END_VC
```

Currently no VC parameters are supported.

HUD block (optional):

HUD mode and parameters. If the HUD block is missing, no HUD is displayed at startup.

```
BEGIN_HUD
        <HUD parameters>
END_HUD
```

<HUD parameters>:

Parameter	Type	Description
TYPE	Flag	`Orbit` | `Surface` | `Docking`

Left/Right MFD blocks (optional):

Left/right MFD type and parameters. If the block is missing, the corresponding MFD is not displayed. Note that custom MFD modes may have their own set of parameters.

```
BEGIN_MFD Left/Right
        <MFD parameters>
END_MFD
```

<MFD parameters>:

Parameter	Type	Description
TYPE	Flag	MFD type: `Orbit|Surface|Map|Launch|Docking|OAlign|OSync|Transfer`
REF	S	Reference object (Orbit and Map MFD only)
TARGET	S	Target object (for Orbit, OAlign and OSync MFD only)
BTARGET	S	Base target (for Map MFD only)
OTARGET	S	Orbit target (for Map MFD only)
PROJ	Flag	`Ecliptic|Ship|Target` (for Orbit MFD only)
MODE	Flag	`Intersect 1|Intersect 2|Sh periapsis|Sh apoapsis|Tg periapsis|Tg apoapsis|Manual axis` (for OSync MFD only)
MANUALREF	F	Reference axis position [deg] (for OSync MFD in manual mode only)
LISTLEN	I	Number or orbit time listings (for OSync MFD only)

Ship list:

List of spacecraft. The list must at least contain the vessel referred to by the `Focus` entry.

```
BEGIN_SHIPS
```

```
        <Ship 0>
        <Ship 1>
        . . .
        <Ship n-1>
END_SHIPS
```

Ship entries <*Ship i*>:

```
<Vessel name>[:<Class name>]
        <Vessel parameters>
END
```

<*Vessel name*>: ship identifier string

<*Class name*>: vessel class (if applicable). If no class is specified, a .cfg file for the vessel, <*vessel name*>.cfg is required.

<*Vessel parameters*>:

Parameter	Type	Description
STATUS	Flag	`Landed` *<planet>* | `Orbiting` *<planet>*
BASE		*<base>*:*<lpad>* (only for STATUS Landed)
HEADING	F	Orientation (only for STATUS Landed)
RPOS	V	Position rel. to reference (only for STATUS Orbiting)
RVEL	V	Velocity rel. to reference (only for STATUS Orbiting)
ELEMENTS	List	Orbital elements. This is an alternative to RPOS and RVEL for vessels with STATUS Orbiting. The list contains 7 entries: semi-major axis *a* [m], eccentricity *e*, inclination *i* [°], longitude of ascending node Ω [°], longitude of periapsis ϖ [°], mean longitude at reference date [°], and reference date in MJD format.
AROT	V	Orientation: rotation angles of object frame (only for STATUS Orbiting)
VROT	V	angular velocity [°/s] (only for STATUS Orbiting)
FUEL	F	Fuel level (0 to 1). This entry sets the level of all propellant resources to the same level. For individual settings, use PRPLEVEL option instead.
PRPLEVEL	List	List of propellant resource levels. Each entry is of the form *<id>*:*<level>*, where *<id>* is the resource identifier, and *<level>* is the propellant resource level (0..1).
THLEVEL	List	List of thruster settings. Each entry is of the form *<id>*:*<level>*, where *<id>* is the thruster identifier (in the order of thruster creation), and *<level>* is the thruster level (0..1). Thrusters with level 0 can be omitted.
DOCKINFO	List	Docking status list. This contains information about all docked vessels. Each entry is of the form *<id>*:*<rid>*,*<rvessel>* where *<id>* is the docking port identifier, *<rid>* is the docking port identifier of the docked vessel, and *<rvessel>* is the name of the docked vessel. Only occupied docking ports are listed. See notes below.

Note that individual vessel types may define additional parameter entries.

Docking vessels

There are two ways to define vessels as being assembled into a superstructure by docking them together:

- Place the vessels so that their docking ports coincide (by using appropriate RPOS, RVEL, AROT and VROT parameters for both). Orbiter will dock two vessels automatically if their docking ports are close enough.

- Define the DOCKINFO lists for both vessels so that they reference each other. Orbiter will then attach the vessels accordingly. **Important:** The RPOS, RVEL, AROT and VROT parameters of the first vessel in the list which belongs to the superstructure are used to initalise the state vectors of the superstructure. All subsequent vessels docked to the same superstructure do not need to define these parameters.

4 Mesh files

Orbiter uses a proprietary mesh file format. Mesh files are ASCII text files. (A binary format may be introduced in the future). Mesh files are located in the `Meshes` subdirectory unless the MeshDir entry in Orbiter.cfg points to a different directory.

Orbiter meshes are defined in a left-handed coordinate system. Vessel meshes should be oriented such that the *vessel's nose* (or more precisely, its *main thrust direction*) points in the positive z-direction, the positive x-axis points *right*, and the positive y-axis points *up*.

The units for vertex coordinates are *meters* [m].

Mesh file format:

`MSHX1`	header
`GROUPS` <*n*>	<*n*>*:* number of groups
<group 1>	group spec 1
<group 2>	group spec 2
…	
<group n>	group spec *n*
`MATERIALS` <*m*>	<*m*>*:* number of materials
<mtrl-name 1>	material name 1
<mtrl-name 2>	material name 2
…	
<mtrl-name m>	material name *m*
<material 1>	material spec 1
<material 2>	material spec 2
…	
<material m>	material spec *m*
`TEXTURES` <*t*>	<*t*>*:* number of textures
<tex-name 1>	texture name 1
<tex-name 2>	texture name 2
…	
<tex-name t>	texture name *t*

Group specs:

[`LABEL` <*label*>]	group label; optional
[`MATERIAL` <*i*>]	material index; optional
[`TEXTURE` <*j*>]	texture index; optional
[`TEXWRAP` <*wrap*>]	texture wrap mode: <*wrap*> = `U` or `V` or `UV`; optional
[`NONORMAL`]	"no normals" flag; see below; optional
`[FLAG` <*f*>`]`	multi-purpose bit-flags; see below; optional
`GEOM` <*nv*> <*nt*>	<*nv*>*:* vertex count, <*nt*>*:* triangle count
<vtx 0>	vertex spec 0
<vtx 1>	vertex spec 1
…	
<vtx nv-1>	vertex spec *nv*-1
<tri 0>	triangle spec 0
<tri 1>	triangle spec 1
…	
<tri nt-1>	triangle spec *nt*-1

Vertex specs:

<*x*> <*y*> <*z*> [<*nx*> <*ny*> <*nz*> [<*tu*> <*tv*>]]

<*x*> <*y*> <*z*>:	vertex position
<*nx*> <*ny*> <*nz*>:	vertex normal (optional)
<*tu*> <*tv*>:	texture coordinates (optional)

Missing normals are automatically calculated as the mean of the normals of adjacent faces. Texture coordinates are only required if the group uses a texture.

Triangle specs:

<i> <j> <k>	vertex indices (zero-based). Left-hand face is rendered.

Material specs:

`MATERIAL` *<mtrl-name>*	material header
<dr> <dg> <db> <da>	Diffuse colour (RGBA)
<ar> <ag> <ab> <aa>	Ambient colour (RGBA)
<sr> <sg> <sb> <sa> <pow>	Specular colour (RGBA) and specular power (float)
<er> <eg> <eb> <ea>	Emissive colour (RGBA)

4.1 Mesh groups

Meshes are divided into groups. Each group can define its own material and texture specification. For example, if you want different parts of the object to have different material properties, you need to split the mesh into groups accordingly.

Each group contains

- An optional label (tag LABEL). The label must be a single word without white spaces. It has no direct effect on the mesh, but can be used to associate a name with a mesh group. Named groups are easier to access from within a vessel module code than group indices (e.g. for defining animations etc.)
- An optional material index. Indices ≥1 select a material of the mesh's material list. Index 0 means "default material" (which is white, diffuse and opaque). If the group doesn't specify a material index it inherits the previous group's material. The first group in the mesh *must* specify a material index, otherwise the result is undefined.
- An optional texture index. Indices ≥1 select a texture from the mesh's texture list. Index 0 means "no texture". If the group doesn't specify a texture index it inherits the previous group's texture. The first group in the mesh *must* specify a texture index, otherwise the result is undefined.
- An optional TEXWRAP flag. This defines how textures wrap around the object. "U" causes textures to wrap in the u-coordinate direction in texel space, "V" wraps in v-coordinate direction, and "UV" wraps in both directions. Default is no wrapping.
- An optional NONORMAL flag. This indicates that vertex definitions in this group don't contain normal definitions, and the first two numbers after the vertex coordinate (x,y,z) triplet is interpreted as texture coordinate (u,v) pair.
- An optional FLAG entry. This allows to specify a user-defined 32-bit flag (in hex format) whose interpretation is context-dependent. Below is a list of flags currently recognised by Orbiter:

Mesh type	Flag	Interpretation
Vessel	0x00000001	Do not use this group to render ground shadows
Vessel	0x00000002	Do not render this group

Vessel	0x00000004	Do not apply lighting when rendering this group
Vessel	0x00000008	Texture blending directive: additive with background

- A GEOM specification, defining the number of vertices and triangles in the group.
- A vertex list (see below)
- A triangle list (see below)

Vertex lists

Each group contains a vertex list, defining the positions, and optionally normal directions and texture coordinates of the vertices in the group.

Each line in the list defines a vertex, and contains up to 8 floating point numbers (separated by spaces)

- The first 3 numbers contain the cartesian vertex coordinates (x,y,z) in the object local coordinate space. Units are meters [m]
- The next 3 numbers (if present) contain the vertex normal direction (nx,ny,nz) (unless the group has set the NONORMAL flag). The normal direction is the direction perpendicular to the mesh surface at the vertex position. Orbiter needs this to generate correct lighting effects. If no normals are specified (or if the NONORMAL flag is set) Orbiter guesses the normal direction as the average of the normals of the surrounding triangles. This works well for smooth surfaces, but should be avoided for surfaces which contain sharp edges. Normal directions should be normalised, i.e. $\mathrm{sqrt}(nx^2+ny^2+nz^2) = 1$.
- The next 2 numbers (if present) contain the vertex texture coordinates (u,v). Texture coordinates are only required if the group uses a texture (i.e. has texture index ≥1). Texture coordinates define how a rectangular 2D texture is mapped onto the object surface. Texture coordinate (0,0) refers to the lower left corner of the texture, (1,1) refers to the upper right corner. Coordinates > 1 are allowed and cause textures to repeat periodically.

Notes:

- Vertices located at sharp edges or corners require multiple entries in the vertex list, because they have multiple normal directions (in other words, the surfaces are *non-differentiable* at edges). In that case you should always define the normals in the mesh file, and not leave it to Orbiter to generate them for you. Otherwise the edges will appear unrealistically smooth.
- Likewise, vertices with multiple vertex coordinates (e.g. at the edge between two texture maps) need multiple entries in the vertex list.

Triangle lists

The group's triangle list follows immediately below the vertex list. It defines the triangles which compose the group's mesh surface.

- Each line in the list defines a triangle and consists of 3 integer numbers (i,j,k). Each of the numbers specifies a vertex from the group's vertex list (starting from 0)

- Only the “clockwise” (CW) side of each triangle is rendered: the side which, if you look at it, has the vertices arranged in a clockwise order. The opposite “counterclockwise” (CCW) side is invisible.
- If you need to render both sides of a triangle (e.g. for a thin plate) you need to define two triangles.
- If you want to flip the rendered side of a triangle (e.g. to correct for “inside out” artefacts) you need to rearrange the triangle indices in the following way: (i,j,k) -> (i,k,j)

4.2 Material list

Materials allow to specify the homogeneous lighting properties of a mesh group. The material lists consists of

- A header line, `MATERIALS <m>`, defining the number <m> of materials.
- A list of material *names*.
- A list of material *properties*.

Each material property specification consists of 4 RGBA quadruplets, where R, G and B define the red, green and blue components, and A is the opacity. RGB values should be between 0 and 1, but can be > 1 for special effects. A *must* be between 0 (fully transparent) and 1 (fully opaque).

- The first line specifies the *diffuse material colour*. This is the colour that is diffusely (in all directions) reflected from an illuminated surface.
- The second line specifies the *ambient material colour*. This is the colour of an unlit surface.
- The third line specifies the specular colour. This is the colour of light reflected by a polished surface into a narrow beam. The *power* entry specifies the width of the cone into which specular light is reflected. Higher values mean a narrower cone, i.e. sharper reflections. Typical values are around 10. If omitted, the default value for power is 0.
- The fourth line specifies the *emissive colour*. This is the colour of light emitted by a glowing surface.

4.3 Texture list

The texture list contains the names of texture files used by the various mesh groups. Texture names should contain file extensions “.dds” but no directory paths. Textures must be located in Orbiter’s `Textures` subdirectory.

Notes:

- Textures must be in DDS format (“Direct Draw Surface”). A DirectX SDK tool, `dxtex`, which is included in the Orbiter SDK package, allows to convert BMP bitmaps into DDS.

- You should store the textures either in DXT1 compressed format (opaque textures or textures with binary transparency), or in DXT5 compressed format (for textures with continuous transparency).
- For maximum compatibility, avoid textures larger than 256x256 pixels, because of limitations of some older graphics cards.
- If a texture is to be dynamically updated during the simulation (e.g. instrument panels in virtual cockpits), the texture name should be followed by the flag 'D'. Orbiter will decompress these textures to allow more efficient dynamic updates.

4.4 Performance optimisation

To achieve the best results with your new mesh, consider the following points:

- Texture groups which use the same texture should be stored in sequence in the mesh. Unnecessary switching between textures can degrade performance if textures must be swapped in and out of video memory.
- Within a sequence of groups using the same textures, groups which use the same material should be stored in sequence. Again, this avoids the need of switching render parameters.
- Avoid large numbers of very small groups. If small groups use the same parameters (material, texture, etc.) they should be merged into a single group.
- Groups which use transparent materials or textures should be sorted to the end of the mesh. If transparent groups overlap, the innermost ones should be listed before the outer ones.

  In order to render transparency correctly, DirectX requires the scene seen through the transparent object to be fully built before the transparent object itself is rendered. Any objects rendered after the transparent object will be masked by it.
- Objects with transparency and specular reflection are more expensive to render than opaque and diffusive objects, so use these features sparingly.
- And most importantly, *keep the vertex count low!* (See section **Error! Reference source not found.**)

4.5 Mesh converters

If you want to convert an existing model into an Orbiter mesh, check the Orbiter web forum for mesh converters created by other users. There is currently a converter which converts from Truespace asc format, which many 3D editors can export. If you have written your own mesh editor or converter, publish it!

4.6 Mesh utilities

The Orbiter SDK contains a few utilities that help to extract data from mesh files. They are located in the Orbitersdk\utils folder.

shipedit: extracts geometric information from a mesh that are useful for defining physical parameters for vessel modules. These include the bounding box extents, volume, cross-sectional areas, and inertia tensor for homogeneous density distribution.

meshc: mesh compiler. Eventually this may be extended to convert mesh files from text to a binary format (for more compact storage and faster loading) but currently it only extracts mesh parameters into a C header file that can be included in a vessel module project for convenient access to named mesh groups.