

Orbiter Flight Recorder and Playback Technical Reference

Martin Schweiger
Thursday, 06 April 2006

Introduction

The purpose of this project is the extension of the standard Orbiter functionality to allow the recording and playback of spacecraft trajectories. The format of the recording streams is public so that external applications such as trajectory optimisation programs can be used to generate the data streams, and to use Orbiter as a visualisation tool for these pre-computed trajectories.

The recorded data include position and velocity samples, attitude data samples, and articulation data which mark events such as engine levels, booster separation, animations, etc. Different data formats (e.g. different frames of reference) are supported to simplify the interfacing with external applications.

Additions to the Orbiter Programming Interface for recording and reading vessel-specific articulation data are provided to enable addon developers to add specific event types in the vessel module code.

Sequence recording

Flight sequences can be recorded and played back later. Currently, recorded data include:

- **Position and velocity.** At the moment, these data are recorded relative to the reference planet, either in a non-rotating reference system (ecliptic and equinox of J2000), or a rotating equatorial reference system. As a result, trajectories are currently recorded in an absolute time frame. Samples are written in regular intervals (currently 4 seconds) or if the velocity vector rotates by more than 5 degrees.
- **Attitude.** Attitude data are saved in terms of the Euler angles of the spacecraft with respect to the ecliptic reference frame or local horizon frame. Samples are written whenever one of the angles has changed by more than a predefined threshold limit.
- **Articulation data.** These include changes in thrust level of individual spacecraft engines, changes in recording speed, onscreen notes, and custom events recorded by individual spacecraft modules, such as animations.

To start recording a flight sequence, launch an Orbiter scenario and start the recorder by pressing Ctrl-C, or from the recorder dialog box (Ctrl-F5). The recording can be stopped by pressing Ctrl-C again or by terminating the simulation. Currently, all spacecraft in the scenario are recorded. Selective recording will be implemented later.

Sequence playback

Each recording generates a new scenario under the "Scenarios\Playback" subdirectory, with the same name as the original scenario. The playback scenario defines the simulation state at the moment when the recording was started. The only difference between standard and playback scenarios is an additional entry "FLIGHTDATA" in each of the recorded spacecraft sections.

Playback scenarios are launched like standard scenarios. On launch, a "Playback" indicator is displayed at the bottom of the simulation window. All spacecraft follow their recorded trajectories until the end of the recording sequence is reached or until playback is terminated by the user with Ctrl-C. At that point, Orbiter's own time propagation mechanism takes over again, and spacecraft return to user control.

Position and attitude data are interpolated between the recorded samples during playback. The recorded articulation events are effective instantaneously.

During playback, the user can manipulate the camera views, switch between camera targets, and operate cockpit instruments such as the MFD displays.

The playback speed (time compression) can either be controlled manually by the user, or set automatically from data tags in the articulation stream.

State interpolation

Orbiter interpolates position and velocity vectors assuming piecewise linear acceleration. Let t_0 and t_1 be two consecutive samples, with recorded state vectors

$$\begin{aligned} r(t_0) &= r_0 & r(t_1) &= r_1 \\ v(t_0) &= v_0 & v(t_1) &= v_1 \end{aligned}$$

To find the interpolated state at time t with $t_0 \leq t \leq t_1$, assume linear acceleration between t_0 and t_1 :

$$a(t) = a_0 + b\Delta t$$

where $\Delta t = t - t_0$. To find a_0 and b which satisfy the boundary conditions, we integrate the state vectors:

$$v(t) = \int_0^{\Delta t} a(t')dt' = v_0 + a_0\Delta t + \frac{1}{2}b\Delta t^2$$

$$r(t) = \int_0^{\Delta t} v(t')dt' = r_0 + v_0\Delta t + \frac{1}{2}a_0\Delta t^2 + \frac{1}{6}b\Delta t^3$$

Substituting the boundary conditions at t_1 and solving for a_0 and b gives

$$a_0 = \frac{2[3(r_1 - r_0) - \Delta T(2v_0 + v_1)]}{\Delta T^2}$$

$$b = \frac{6[2(r_0 - r_1) + \Delta T(v_0 + v_1)]}{\Delta T^3}$$

with $\Delta T = t_1 - t_0$.

Attitude interpolation

Currently, spacecraft orientations are interpolated linearly between attitude samples, resulting in piecewise constant angular velocities. The interpolation is implemented by transforming the recorded Euler angle data into a quaternion representation, and performing a spherical interpolation between pairs of quaternion samples.

The attitude data stream should provide sufficiently dense sampling so that noticeable jumps in angular velocity are avoided.

Orbiter's built-in recording module writes a sample to the attitude stream

- when the orientation has changed by more than 0.06 rad since the last sample, or
- when the orientation has changed by more than 0.001 rad and no sample has been written for more than 0.5 seconds.

File formats

All flight data are recorded under the "Flights" subdirectory. Each recording generates a new subdirectory with the name of the scenario. If the directory already exists, it is overwritten.

Each recorded object generates three files, where <object name> is the name of the vessel as defined in the scenario file:

<object name>.pos

Position and velocity data relative to a reference body. Each line contains either a data sample, or a format directive. The following directives are currently supported:

STARTMJD <mjd>

Defines an absolute time reference for the simulation start time. <mjd> is a floating point number defining the start date in MJD (Modified Julian Date) format. Only a single STARTMJD tag should be provided at the beginning of the stream. Note that this value is currently not used by the simulator, because the simulation start date is read from the corresponding scenario file.

REF <reference name>

Defines the reference object (planet, moon, sun) relative to which the following data samples are calculated. Whenever the trajectory enters the sphere of influence of a different object, another "REF" line should be added, and the following data samples computed with respect to the new object. <reference name> must correspond to a celestial object defined in the Orbiter planetary system. There is no default object, therefore a REF directive must appear at the beginning of the file before any data samples.

FRM [ECLIPTIC | EQUATORIAL]

Orientation of the reference frame for all following data samples. This can be either the ecliptic of the J2000 epoch, or the (rotating) equatorial frame of the reference body. The default setting is "ECLIPTIC".

CRD [CARTESIAN | POLAR]

Coordinate and velocity data format for all following data samples. This can be either in rectangular cartesian format (x, y, z) [m] and $(\dot{x}, \dot{y}, \dot{z})$ [m/s], respectively, or in spherical polar coordinates (r, ϕ, θ) [m, rad, rad] and

$(\dot{r}, \dot{\phi}, \dot{\theta})$ [m/s, rad/s, rad/s], respectively, with radial distance r , polar angle ϕ and azimuth angle θ . If an equatorial frame is selected, ϕ and θ define equatorial longitude and latitude. See Appendix 2 for transformation conventions.

The default setting is “CARTESIAN”.

Any lines not containing one of the above directives are assumed to contain data samples, in the following format:

<SimT> <position> <velocity>

where :

<SimT> is the time since scenario start [s]
 <position> is the object position w.r.t. the current reference object. Depending on the current “CRD” setting, this is a triplet of <x> <y> <z> cartesian coordinates [m], or a triplet of <radius> [m] <longitude> <latitude> [rad], in either ecliptic or equatorial reference frame.
 <velocity> is the object velocity w.r.t. the current reference object. Depending on the current “CRD” setting, this is a triplet of <vx> <vy> <vz> rates in cartesian coordinates [m/s], or a triplet of <radial velocity> [m/s] <longitude rate> <latitude rate> [rad/s], in either ecliptic or equatorial reference frame.

<object name>.att

Attitude data. Each line contains either a data sample, or a format directive. The following directive is currently supported:

STARTMJD <mjd>

Defines an absolute time reference for the simulation start time. <mjd> is a floating point number defining the start date in MJD (Modified Julian Date) format. Only a single STARTMJD tag should be provided at the beginning of the stream. Note that this value is currently not used by the simulator, because the simulation start date is read from the corresponding scenario file.

FRM [ECLIPTIC | HORIZON]

Orientation of the reference frame for all following attitude data samples. This can either be the ecliptic of the J2000 epoch, or the local horizon of the current vessel position. The default setting is "ECLIPTIC".

REF <reference name>

Defines the reference object (planet, moon, sun) relative to which the following data samples are calculated. Note that a REF tag is required after each FRM HORIZON tag, to define the reference object for local horizon calculations, but is optional after a FRM ECLIPTIC tag, because the reference frame is globally fixed. When using FRM HORIZON, the reference should usually be switched by inserting a new REF tag whenever the trajectory enters the sphere of influence of a different object.

Any lines not containing a directive are assumed to contain samples , in the following format:

<SimT> < α > < β > < γ >

where:

<SimT> is the time since scenario start (seconds)
 < α > < β > < γ > are the Euler angles of the local spacecraft frame with respect to the orientation of the reference frame.

Definition: Let

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

be the rotation matrix that transforms from vessel frame to the reference frame (ecliptic or local horizon).

For the ecliptic frame, Orbiter uses the following set of Euler angles:

$$\alpha = \arctan \frac{r_{23}}{r_{33}}, \quad \beta = -\arcsin r_{13}, \quad \gamma = \arctan \frac{r_{12}}{r_{11}}$$

For the local horizon frame, Orbiter uses a different set of Euler angles:

$$\alpha = \arctan \frac{r_{12}}{r_{22}}, \quad \beta = \arcsin r_{32}, \quad \gamma = \arctan \frac{r_{31}}{r_{33}}$$

so that the Euler angles can be directly defined with the bank, pitch and yaw angles of the vessel in the local horizon frame:

Bank angle α : angle between the projection of the horizon normal into the vessel xy-plane, and the vessel "up" direction (0,1,0).

Pitch angle β : angle between the projection of the horizon normal into the vessel yz-plane, and the vessel "up" direction (0,1,0).

Yaw angle γ : angle between the projection of the vessel "forward" direction (0,0,1) into the local horizon plane, and the horizon "north" direction.

<object name>.atc

Articulation data. Each line contains a sample as follows:

<SimT> <id> <data>

where:

<SimT> is the time since scenario start (seconds)

<id> identifies the event type. The generic event types currently supported by Orbiter are listed below. In addition, vessel-specific event types can be directly implemented by the vessel module.

<data> Event-type specific data.

Engine events

Engine events are recorded with the "ENG" tag in the articulation stream. Engine event data consist of

<engine id>:<level>

pairs, where <engine id> is either a zero-based integer index identifying the engine, or an engine group identifier string. <level> is the thrust level (range: 0-1). Integer engine identifiers can usually be obtained from the spacecraft DLL implementation. If groups of engines must be operated simultaneously, it is often more convenient to use group identifiers. The following group labels are supported:

MAIN
RETRO
HOVER
RCS_PITCHUP
RCS_PITCHDOWN
RCS_YAWLEFT
RCS_YAWRIGHT
RCS_BANKLEFT
RCS_BANKRIGHT
RCS_RIGHT
RCS_LEFT
RCS_UP
RCS_DOWN
RCS_FORWARD
RCS_BACK

How the engines are assembled in these groups depends on the vessel module code. Note that not all vessel types may support all of the logical groups listed above. Further, some engines may not be members of any group and therefore must be addressed by their individual integer id's.

Not all engine levels need to be recorded with each sample, but the first and last entries of the file should contain all engines, to provide a fully defined initial and final state.

Other generic events

The following default event tags in the articulation stream are currently recognised by vessels in Orbiter, and written to the atc stream during recording:

RCSMODE <mode>	Switch Reaction Control System mode to <mode>, where <mode> is an integer in the range 0...2. See the RCS_XXX constants defined in OrbiterSDK/include/OrbiterAPI.h for a list of supported RCS modes.
ADCMODE <mode>	Switch aerodynamic control mode to <mode>
NAVMODE <mode>	Switch autonav mode to <mode>, where <mode> is an integer in the range 1...7. See the NAVMODE_XXX constants defined in OrbiterSDK/include/OrbiterAPI.h for a list of supported nav modes. Note that some modes are exclusive, i.e. setting one may implicitly clear another mode.
NAVMODECLR <mode>	Explicitly clear autonav mode <mode>.
UNDOCK <dock>	Undock vessel attached at port <dock>
TACC <acc> [<delay>]	Set time acceleration factor to <acc>. This tag is only written if the “Record time acceleration” option is enabled in the recorder dialog (Ctrl-F5). If the optional <delay> value is provided, the change in time acceleration is non-instantaneous. Instead, time acceleration changes by one order of magnitude per <delay> seconds.

Onscreen annotations during playback

Playback sequences can be annotated with onscreen messages. The messages appear on top of the render window at the time defined in the playback stream. Some basic formatting (position, size and colour) are available.

Annotations can be added by inserting NOTE tags manually into the articulation stream after the recording has been completed. The format is

<SimT> NOTE <text>

where <text> is the text of the note. The text must be entered as a single line, but will be displayed in multiple lines on screen as required. The notes must be sorted appropriately into the stream, so that all <SimT> tags appear in increasing order. The text remains visible until it is replaced by a new note, or until it is explicitly removed with

<SimT> NOTEOFF

or until the end of the replay sequence. Note that annotations can be displayed by all vessels in the playback scenario. However, in general notes should only be generated by the focus vessel to avoid confusion.

The following formatting tags are available:

<SimT> NOTEPOS <x1> <y1> <x2> <y2>

defines the position of the rectangle where the note appears. All values are fractions of the simulation window size, in the range 0...1. <x1> and <x2> are the left and right edges of the note rectangle, <y1> and <y2> are the top and bottom edges. $x1 < x2$ and $y1 < y2$ are required. If the rectangle is set too small, part of the note may not be displayed.

<SimT> NOTESIZE <scale>

defines the size scale for the note text, where 1 denotes the default size. The actual text size is scaled with the simulation window size.

<SimT> NOTECOL <r> <g>

defines the colour of the note text, where <r> <g> are the red, green and blue components, respectively. Values must be in the range 0...1.

Recording and playback of vessel-specific events

Using Orbiter's Application Programming Interface (API) it is possible to record and play back events that are not recognised by the Orbiter core. These may include animations like lowering or retracting landing gear, opening cargo bay doors, separating booster rockets, etc.

The API interface for recording and playback consists of two functions:

```
void VESSEL::RecordEvent (const char *event_type, const char *event) const
```

The vessel calls this function to record an event to the articulation stream. The record consists of an event type identifier (*event_type*) and the event itself (*event*). *event_type* must be a single word (no whitespace), while *event* can contain multiple items separated by spaces. Orbiter only writes the event if a recording session is active. Otherwise the function call is ignored. The event appears in the stream in the format

```
<SimT> <event_type> <event>
```

During a playback session, any events that are read from the articulation stream but contain an identifier not recognised by the Orbiter core, are passed on to the vessel module via the callback function

```
virtual bool VESSEL2::clbkPlaybackEvent (double simt, double event_t, const char *event_type, const char *event)
```

where *simt* is the current simulation time, *event_t* is the time recorded with the event, *event_type* is the identifier for the event, and *event* is the recorded event data. An event is processed whenever the simulation time has moved past the recorded time stamp, therefore $simt \geq event_t$.

Examples for vessel-specific custom articulation stream commands can be found in the source code of the "Delta-glider": `Orbitersdk/samples/DeltaGlider/DeltaGlider.cpp`

Examples

Some recorded examples are provided to demonstrate the playback features in orbiter, and the data stream formats. The examples can be found in the Playback scenario folder, and are executed by running Orbiter with the appropriate scenario. The corresponding data streams can be found under the Flights folder.

Glider launch 1:

Scenario:	Glider takeoff and landing.
Attitude stream:	Ecliptic frame of reference
Pos/vel stream:	Geocentric cartesian coordinates in ecliptic frame of reference
Articulation stream:	Exhaust rendering, animations (landing gear, airbrakes)

Glider launch 2:

Scenario:	Glider takeoff and landing.
Attitude stream:	Local horizon frame of reference.
Pos/vel stream:	Geocentric polar coordinates in equatorial frame of reference
Articulation stream:	Exhaust rendering, animations (landing gear, airbrakes)
Notes:	Demonstrates use of horizon frame of reference, where attitude data are provided in terms of bank, pitch and yaw angle of local horizon plane.

Glider in orbit 1:

Scenario:	Glider attitude control in orbit
Attitude stream:	Ecliptic frame of reference
Pos/vel stream:	Geocentric cartesian coordinates in ecliptic frame of reference
Articulation stream:	Exhaust rendering of reaction control thrusters
Notes:	Demonstrates use of attitude stream for object rotation, including visual cues (RCS thruster rendering).

Glider in orbit 2:

Scenario:	Glider in orbit: demonstration of custom animations
Attitude stream:	Local horizon frame of reference
Pos/vel stream:	Geocentric polar coordinates in equatorial frame of reference
Articulation stream:	Custom animation sequences

Notes: This example shows the use of custom animation commands (defined in the vessel plugin module) read from the articulation stream. The glider defines animation commands for various mesh elements (landing gear, airbrakes, airlocks and hatches, radiator deployment, etc.)

Atlantis launch:

Scenario: Space Shuttle launch and orbit insertion.
Attitude stream: Ecliptic frame of reference
Pos/vel stream: Geocentric polar coordinates in equatorial frame of reference
Articulation stream: SRB booster/SSME and RCS thruster exhaust animation, SRB and ET jettison commands
Notes: Shows the use of custom-defined jettison commands for animation control purposes. Articulation stream also has examples for addressing thruster groups with symbolic labels ("ENG MAIN").

Atlantis undocking:

Scenario: Space Shuttle moving away from International Space Station, cargo doors closing.
Attitude stream: Local horizon frame of reference
Pos/vel stream: Geocentric polar coordinates in equatorial frame of reference
Articulation stream: RCS thruster control, cargo bay animation
Notes: Shows custom animation commands defined in plugin module (Atlantis.dll): Radiator and payload bay closing.

Atlantis final approach:

Scenario: Space Shuttle touchdown sequence.
Attitude stream: Ecliptic frame of reference
Pos/vel stream: Geocentric polar coordinates in equatorial frame of reference
Articulation stream: Custom animation sequences
Notes: Shows custom animation commands defined in plugin module (Atlantis.dll): Split rudder brake deployment and landing gear animation. Demonstrates onscreen annotation.

Lunar transfer:

Scenario: Complete transfer simulation from Earth surface (KSC) to landing on the Moon's surface.
Attitude stream: Ecliptic frame of reference
Pos/vel stream: Equatorial frame of reference Earth/Sun/Moon
Articulation stream: Engine events, custom animation, annotations and time acceleration
Notes: Demonstrates sampling density variations by changing the simulation speed during recording, playback time acceleration and onscreen annotation features. Shows transition of reference object in the .pos stream. Demonstrates ability of long playback sequences (~1 hour at recording speed).

Appendix 1: Orbiter reference frames

Orbiter uses *left-handed* coordinate systems throughout.

The global frame.

The global frame of reference is the barycentric ecliptic frame for ecliptic and equinox of epoch J2000, where

+x points to the vernal equinox
+y points to ecliptic zenith
 $\hat{z} = \hat{y} \times \hat{x}$

Rotating planet frames.

Planet frames are fixed to rotating planets, where

+x points from the planet centre to surface point latitude 0, longitude 0.
+y points from the planet centre to the north pole
 $\hat{z} = \hat{y} \times \hat{x}$

Local horizon frames.

Given planetocentric equatorial longitude and latitude (ϕ, θ), the local horizon frame is given by the tangent plane to the (spherical) planet surface, where

+x points east
+y points up

+z points north

Local spacecraft frames.

The orientation of the spacecraft frame is largely up to the designer. By convention, the following is usually adopted:

+x points “right”

+y points “up”

+z points “forward” (in the direction of the thrust vector of the main engines)

Appendix 2: Cartesian and polar coordinates

Given Orbiter’s left-handed coordinate system, the transformation between cartesian positions (x, y, z) and velocities $(\dot{x}, \dot{y}, \dot{z})$ and spherical polar coordinates (r, ϕ, θ) and velocities $(\dot{r}, \dot{\phi}, \dot{\theta})$ is defined as

$$x = r \cos \phi \cos \theta \quad \dot{x} = \dot{r} \cos \phi \cos \theta - r \dot{\phi} \sin \phi \cos \theta - r \dot{\theta} \cos \phi \sin \theta$$

$$y = r \sin \theta \quad \dot{y} = \dot{r} \sin \theta + r \dot{\theta} \cos \theta$$

$$z = r \sin \phi \sin \theta \quad \dot{z} = \dot{r} \sin \phi \cos \theta + r \dot{\phi} \cos \phi \cos \theta - r \dot{\theta} \sin \phi \sin \theta$$

and

$$r = \sqrt{x^2 + y^2 + z^2} \quad \dot{r} = \dot{y} \sin \theta + \cos \theta (\dot{x} \cos \phi + \dot{z} \sin \phi)$$

$$\phi = \arctan \frac{z}{x} \quad \dot{\phi} = \frac{\dot{z} \cos \phi - \dot{x} \sin \phi}{r \cos \theta}$$

$$\theta = \arcsin \frac{y}{r} \quad \dot{\theta} = \frac{\dot{y} \cos \theta - \sin \theta (\dot{x} \cos \phi + \dot{z} \sin \phi)}{r}$$