Universal Cargo System for Orbiter

###### Universal Cargo System for Orbiter (UCSO) is a free, open-source cargo management system for the Orbiter space simulator. It’s designed to be an Orbiter 2016 alternative of UCGO. It provides a rich API for add-on developers to support UCSO in their creations.

## Installation and configuration

To install UCSO, unpack the archive file directly into the Orbiter installation folder. You can also use [Orbiter Add-ons Manager](https://www.orbithangar.com/showAddon.php?id=48079dea-b257-42b1-a10d-b89de58add5e) which will manage all your Orbiter add-ons.

The configuration file is ‘Config\UCSO\_Config.cfg’. Open it with Notepad and edit the options. All options are explained in the file.

## Cargo types

There are 4 cargo types in UCSO:

Static cargo:

Resource cargo:

A static cargo can’t be unpacked or used by vessels. It’s just a mesh. This type example is (CargoContainer).

A resource cargo contains one resource which can be used by vessels, such as fuel or oxygen. It can’t be unpacked. This type example is fuel cargo (CargoFuel).

Unpackable cargo: An unpackable cargo can be unpacked by vessels. It is 2 types:

1. A packable and unpackble cargo: This cargo can be packed and unpacked multiple times.
2. An unpackable only cargo: This cargo can be unpacked one time only, and can’t be packed again.

After unpacking, it can be:

1. A UCSO resource, which can be used by vessels. This type example is (CargoFuelTank).
2. A UCSO module, which can be a breathable or static module. This type examples are the flags, solar panel, table and chairs, and the life module cargoes (CargoFlagXX, CargoSolarPanel, CargoTableChairs, and CargoLifeModule).
3. An independent Orbiter vessel. This type example is ShuttlePB cargo (CargoShuttlePB).

Custom cargo:

These are independent Orbiter vessels that use the custom cargo API to act like normal cargoes. They can be any type as above or even a custom type. This type example is the lamp cargo (CargoLamp) which spawns a lamp that emits light when unpacked

## Standard resource names

These standard resource names should be used by vessel and cargo authors. ***The name must be lowercase***, and spaces can be included.

fuel, ramjet fuel, oxygen, food, water, hydrogen, helium

Note: for all ramjet engine variations, you can drain the normal fuel or ‘ramjet fuel’ (i.e. if the vessel has SCRAM engines, use ‘ramjet fuel’, not ‘scram fuel’).

## XR-2 cargoes platform

UCSO comes with cargoes platform for XR-2, which allows carrying up to 6 UCSO cargoes in XR-2 bay. It occupies the whole bay, so other payloads can’t be carried with the platform.

[Dynamic XRSound](https://www.orbithangar.com/showAddon.php?id=5376bb58-c52b-4708-a4eb-cdcb7eb1dc55) is required to enable the sound support. The platform will work without it, but without any sound.

To add the platform, open the XR-2 payload editor, choose ‘XR2\_UCSOPlatform’ from the payloads list, and add it to the second slot. Switch the active vessel to it via ‘Select spacecraft’ (F3) window by clicking on the XR-2 vessel name and choosing the platform under it. The keyboard shortcuts are shown on the HUD.

The platform has a realism mode, which is enabled by default. In this mode, cargoes can’t be added, grappled, released, and deleted if the bay doors are closed, the SCRAM engine fuel can’t be fed from normal fuel (i.e. the cargo resource type must be ‘ramjet fuel’), and the total cargo mass can’t exceed 10.8 metric tons.

To enable or disable it, open the configuration file: ‘Config\Vessels\XR2\_UCSOPlatform.cfg’, and set ‘RealismMode’ option as required.

Note: In XR-2 configuration file, ***‘OrbiterAutoRefuelingEnabled’ option must be enabled*** to allow draining fuel to the vessel.

Currently, there is no way to drain APU fuel or LOX.

## Stations

UCSO Stations are independent Orbiter vessels that can provide an infinite amount of multiple resources to vessels.

To create a station, open the vessel you want to be a station configuration file which can be found in the ‘Config\Vessels’ folder, and find the attachment sections.

BEGIN\_ATTACHMENT

………………………………………

END\_ATTACHMENT

If you found it, add the UCSO station attachment before END\_ATTACHMENT line:

BEGIN\_ATTACHMENT

………………………………………

P 0 0 0 0 0 1 0 1 0 UCSO\_ST

END\_ATTACHMENT

If you didn’t find it, add the following lines to the end of the file:

BEGIN\_ATTACHMENT

P 0 0 0 0 0 1 0 1 0 UCSO\_ST

END\_ATTACHMENT

At the top of the file, add the resource names separated by commas ***without spaces before the commas*** as following:

UCSO\_Resources = fuel,ramjet fuel,oxygen

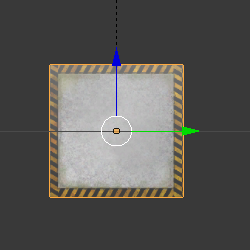
The station can have one or multiple resources. See the standard resource name above.

Save the file and try the station with a vessel that supports UCSO in Orbiter.

## Cargo creation

###### Modelling

The packed cargo bounding box dimensions must be 1.3mx1.3mx1.3m to follow the UCSO standard size. If not followed, the cargo won’t fit into cargo containers on vessels, and the touchdown points won’t be calculated correctly. The bounding box bottom must be at -0.65m, so the cargo is in the middle. You can use the default containers which come with UCSO, so you can work on other things.



The packed cargo polygons should be as low as possible; because vessels can carry a lot of cargoes (e.g. XR-5 can carry up to 300 cargoes). Since a lot of polygons would kill the frame rate, the polygons shouldn’t exceed 200 polygons with a 512px512p texture.

The cargo mesh file must be saved in ‘Meshes\UCSO’ folder, and texture in ‘Textures\UCSO’ folder. The file name must be unique to avoid conflict with other cargoes; so, add letters and numbers in the cargo name (e.g. CargoFuelAR51.msh).

For unpacked cargoes, the polygons shouldn’t exceed 1000 polygons with a 1024px1024p texture. Save the mesh and texture files as above.

###### Configuration file

The cargo configuration file must be saved in the ‘Config\Vessels\UCSO’ folder. The easiest way to make a configuration file is to copy a similar cargo configuration file and edit it. You should check multiple files to see the different options. The options are explained in the configuration files.

The configuration file name must be unique too. The mesh, texture, and configuration file should have the same name, although not necessary.

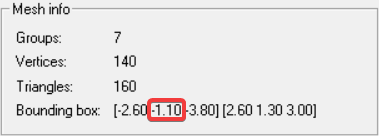
If one of the required (not optional) options is missing, Orbiter will crash when the cargo is loaded with a runtime error and an error message in the log with the missing option.

The cargo mass should be the actual cargo mass, not including the container mass. The container mass is set by UCSO and can be changed in the UCSO configuration file.

If the cargo contains a resource, see the standard resource names above.

For unpackable cargoes, the unpacked height (and the spawn height) is the mesh's lowest point. The cargo height will be set to this height so that the cargo bottom is at the ground level. To get it, use the Shipedit tool in the ‘Orbitersdk\utils’ folder.

Open the cargo unpacked mesh (or the vessel you want to spawn) in the tool and see the mesh bounding box. The mesh's lowest point is the negative Y value (the center value in the negative group).

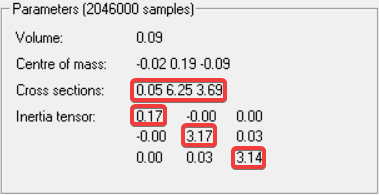


The spawn height should be a positive value, so the spawn height for the picture above is 1.1m

The unpacked size is the cargo unpacked mesh mean radius which is simply the absolute value of the biggest number in the bounding box, so the size for the picture is 3.8m.

For the unpacked attachment point position, see the attachment points section below.

The unpacked PMI and CS are the inertia tensor and the cross-sections of the mesh. To get it, click on ‘Calc -> Start/continue MC integration’. Wait until the numbers at the ‘Parameters’ section are stable (It should be around 500,000 samples), then stop it from ‘Calc -> Stop MC integration’.



The cross-sections should be separated with spaces, so the cross-sections for the picture is ‘0.05 6.25 3.69’. The PMI is diagonal as marked in the picture, so the PMI is ‘0.17 3.17 3.14’.

## Cargo texturing

Creating a skin or texture for cargo is basically creating a new cargo with a different texture.

To texture a cargo, make a copy of its configuration file in ‘Config\Vessels\UCSO’ folder and rename it with a unique name as detailed earlier.

Open the configuration file and find the mesh file name. If you are making a texture for the packed cargo, the mesh file name is the ‘PackedMesh’ value. If you are making for the unpacked cargo, the mesh file name is the ‘UnpackedMesh’ value.

Make a copy with a unique name of the mesh file which can be found in ‘Meshes\UCSO’ folder and scroll down until you find the textures section:

TEXTURES 1

UCSO\LifeModule.dds

The recommended texture size is 512px512p for packed meshes, and 1024px1024p for unpacked meshes.

After finishing, change the texture file in your mesh file to your texture file, and the mesh file in your configuration file.

## Attachment points

The attachment point for any packed UCSO cargo is at the center of the cargo bottom. If you are making a custom cargo or adding USCO support for a vessel, keep this in mind. The attachment point coordinates in the Orbiter left-handed coordination system is (X: 0, Y: -0.65, Z: 0), the direction is (X: 0, Y: -1, Z: 0), and the rotation is (X: 0, Y: 0, Z: 1).

For unpacked cargoes, the EVA astronaut will carry the cargo from the unpacked attachment position. It should be set at the lowest point of the mesh ‘front’, so the cargo is carried from the bottom.

The attachment point rotation and direction must be set properly, to have the texture visible correctly and the release velocity if released in space applied in the correct direction.

For example, if your attachment point is below the vessel, the direction and rotation must be set inverted to have the velocity applied downwards. Otherwise, the cargo might go sideward or through the vessel.

If your vessel has multiple slots, leave at least 1.7m between each cargo attachment point to have 0.2m between each cargo edge.

If your vessel is an astronaut, the attachment point Z axis should be reversed, so the attached cargo front faces the astronaut front (i.e. rotation (X: 0, Y: -1, Z: 0) and direction (X: 0, Y: 0, Z: -1)).

A very handy tool is [Mesh Wizard](https://www.orbithangar.com/searchid.php?ID=2740), which will show the attachment point axes. In order to run the program, you need to download [MSVBVM50.DLL](https://www.dll-files.com/msvbvm50.dll.html) and place it in its folder, open the command prompt as an administrator in its folder, and execute the following commands:

regsvr32 COMCTL32.OCX

regsvr32 MSFLXGRD.OCX

Open it, select ‘Calculator -> Calculate ROT vector’, and enter your attachment point data. A normal upward attachment data is DIR (X:0, Y:0, Z:1) and ROT (X:0, Y:1, Z:0).

The XY or the rear view blue line is the cargo release direction for space release (e.g. if it points up the cargo will go upward, if it points right the cargo will go right, etc.).

Mesh Wizard can be used also to set the attachment point position. Open your mesh from ‘File -> Load’ and set the attachment point position in ‘Center view coord’. Move between different views using the DeltaGlider buttons.

## Custom cargoes

A custom cargo is a normal Orbiter vessel that uses the API to act like normal cargoes.

Follow the instructions and restrictions in the cargo creation section above (e.g. packed dimensions must be 1.3m, etc.). You can save the meshes and textures anywhere you want, but it’s recommended to save them in UCSO folders with unique names.

The configuration file must be saved in ‘Config\Vessels\UCSO’ folder, and its name must begin with ‘CargoCustom’ and should be unique (e.g. CargoCustomFuelAR51.cfg). Fill the configuration file as any Orbiter vessel configuration file.

## API

To make a custom cargo or support UCSO in your vessel, you need to use the C++ API.

You need to set your project to import the Orbiter property sheet. In Visual Studio, select ‘View -> Other Windows -> Property Manager’. The property manager should be visible on the right panel. Open it, right-click on your project and select ‘Add Existing Property Sheet’. Go to the ‘Orbitersdk\resources’ folder and select ‘Orbiter.props’ file. The Orbiter property sheet should be added to your project now.

You need to link against the API library (UCSO\_API.lib file). Open your project properties by right-clicking on your project and selecting Properties. Go to ‘Linker -> Input’. In the ‘Additional Dependencies’ field, add ‘UCSO\_API.lib’ with a semicolon after ‘orbitersdk.lib.



Note that the library can be used only in release configuration. If you want a debug version, you must build UCSO source code in debug configuration.

If you are making a custom cargo, include ‘UCSO\_CustomCargo.h’ file. The API depends on Orbiter SDK, STL, and UCSO. If UCSO isn’t installed and a custom cargo is loaded, Orbiter will crash with a runtime error and an error message in the Orbiter log.

To set the cargo attachment points correctly, see the attachment points section above.

For vessels, include ‘UCSO\_Vessel.h’ file. The API doesn’t depend on UCSO, so it’ll work without it. A warning message will be written in the Orbiter log.

You can know if UCSO is installed by calling the GetUCSOVersion method, which will return a null pointer if UCSO isn’t installed.

To set the cargo attachment points correctly, see the attachment points section above.

See the custom cargo and vessel API walkthrough documents in the ‘Orbitersdk\doc’ folder.

## Credits

[Fred18](https://www.orbiter-forum.com/member.php?u=8871): The ground release rotation and touchdown points code.

[Woo482](https://www.orbiter-forum.com/member.php?u=195): The ground release location code, and the XR2 cargoes platform 3D model.

[Gattispilot](https://www.orbiter-forum.com/member.php?u=27): The flags 3D model.

[Hasnat Ahmed Khan](https://sketchfab.com/3d-models/container-92bd84031ebc4ddcbf3b3d3689c4bf31): The container 1 3D model. The model was modified.

[da3dalus](https://sketchfab.com/3d-models/crate-5f4a1c3655e7430b9cb2d919fb6b760a): The container 2 3D model. The model was modified.

[forest\_cat](https://sketchfab.com/3d-models/sci-fi-props-1-cbd970aafa9d468994af03a6b7fa0017): The container 3 3D model. The model was modified.

[Fabian van Dorst](https://sketchfab.com/3d-models/solar-panel-ff765abe2d324c91899541b43cc40c72): The solar panel 3D model. The model was modified.

[toAflame](https://sketchfab.com/3d-models/wooden-chair-55153fc8b04143ad8f19fdb48b8061af): The wooden chairs and table 3D model. The model was modified.

[Robin Butler](https://sketchfab.com/3d-models/stylized-low-poly-sci-fi-buildings-701381fef32444e79ad804315e563049): The life module and lamp 3D models. The models were modified.

[Ofiusa3D](https://sketchfab.com/3d-models/gas-tank-5eb9dc078f4e4210b0a693056844f4c5): The fuel tank 3D model. The model was modified.

## About

UCSO is free and open source under the GPLv3 license. The source code can be found in the [UCSO GitHub repository](https://github.com/abdullah-radwan/UCSO). All contributions are appreciated.

Copyright © [Abdullah Radwan](mailto:abbodmar@gmail.com)