Universal Cargo System for Orbiter

## Universal Cargo System for Orbiter (UCSO) is a cargo management system for Orbiter 2016, designed to replace UCGO. It provides an API for vessel authors to add UCSO support to their vessels.

Installation

### Simply unpack the ZIP file into the Orbiter root folder. You can use [Orbiter Addons Manager](https://www.orbithangar.com/searchid.php?ID=7213) which I developed to manage all of your add-ons, not just UCSO.

### You need [ShuttlePB UCSO](https://github.com/abdullah-radwan/ShuttlePB_UCSO/releases) to run the scenarios. Make sure to match the version.

Configuration

### The UCSO configuration file is ‘Config\UCSO\_Config.cfg’. You can open it with Notepad. All options are explained there.

Standard keyboard shortcuts

|  |  |
| --- | --- |
| Key | Function |
| Shift + A | Add cargo |
| Shift + G | Grapple cargo |
| Shift + R | Release cargo |
| Shift + P | Pack cargo |
| Shift + U | Unpack cargo |
| Shift + S | Use resource cargo |
| Shift + D | Delete cargo |

### Note that not all function might be implemented in the vessel, and the keys might be different. Review the vessel manual for details.

Cargoes type

### There are 3 types of cargoes in UCSO: Static, resource and unpackable cargoes. There is one example for each cargo in the default distribution.

### A static cargo, as its name implies, is static. It can’t be packed, unpacked, or used by vessels. The example of this type is ‘CargoContainer’ cargo.

### A resource cargo contains a resource which can be used by vessels if the vessel supports.

### The example cargo for this is ‘CargoFuel’ cargo, which contains 1000 kilograms of fuel.

### An unpackable cargo is a cargo which can be unpacked. When unpacked, it can be a UCSO module or an Orbiter vessel. UCSO modules can be packed again, while Orbiter vessels can’t be packed.

### UCSO module examples are ‘CargoSolarPanel’ and ‘CargoTableChair’ cargo, which spawns a solar panel and a table with 4 chairs, respectively.

### There are 3 unpack modes for Orbiter vessels: ’Landed’, ‘Delayed’, and ‘Manual’.

### Landed means the cargo will be automatically unpacked when it touches the ground. Delayed means the cargo will be unpacked when it reaches the set delay. Manual means the cargo has to be unpacked manually by an object (e.g. a vessel).

### An Orbiter vessel example is ‘CargoShuttlePB’ cargo, which spawns a ShuttlePB vessel after 25 seconds.

Cargo creation

### No programming is required to make a cargo.

### To create a cargo, you need to create a configuration file in the ‘Config\Vessels\UCSO’ folder. Copy the example for the cargo type you want, then modify it.

### For the mesh, you can use the default mesh, which is ‘UCSO\UCSO\_Cargo’. If you want to create a mesh, the following restrictions must be followed.

### The mesh bounding box must be 1.3mx1.3mx1.3m, and the mesh bottom must be at -0.65 meter. The polygons should be as low as possible, as the cargoes can be massive (e.g. XR-05 can carry more than 300 cargoes).

### For a 1000 polygons mesh, the total polygons can be as high as 300,000 polygons! This is a frame rate killer. The recommended polygons are between 50 and 200, with one 512x512 or 1024x1024 texture.

### After finishing, place the mesh in ‘Meshes\UCSO’ folder, and the texture in ‘Textures\UCSO’ folder, then modify the configuration file accordingly.

### For the mass, UCSO will automatically add the container weight for the cargo (The default is 85 kilograms, which can be modified in UCSO configuration file).

### The configuration, mesh, and texture filenames must be unique, to avoid conflicts with other add-ons. For example, if your cargo is fuel and your name is Suzan, a good name is SZ12CargoFuel.

### If you made a mistake, and one of the main configuration file entries are misssing, Orbiter will crash with a runtime error, and the missing entry will be written in ‘Orbiter.log’ file.

### If you’re making a resource cargo, it’s recommended to use standard keywords below, so the vessel authors can use it also:

#### Fuel, Oxygen, SCRAM Fuel, APU Fuel, Food, Water, Hydrogen.

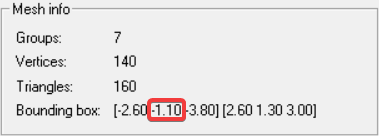
### The resource types aren’t case-sensitive, and white spaces can be included.

### If you’re making an Orbiter vessel unpackable cargo, here are some notes:

### UCSO will add numbers to the spawn name if there are vessels with the same name.

### The spawn module is the vessel configuration file path as viewed from ‘Config\Vessels’ folder without ‘.cfg’. For example, DeltaGlider path is ‘Deltaglider’.

### The spawn height is the vessel height if released on the ground. To get it, open the vessel mesh with ‘Shipedit’ tool which can be found in ‘Orbitersdk\utils’ folder. The height is the absolute value for the negative value in the center in ‘Bounding box’. For ShuttlePB, this value is 1.10. You might want to increase the height a little bit, as Orbiter rendering isn’t perfect, and the vessel might appear below the surface.



### If you are making a UCSO module unpackable cargo, set the unpacked height as detailed above.

### The unpacked size is simply half of the unpacked mesh longest dimension. For the ShuttlePB example above, the size is the absolute value of the largest number, which is 3.8m.

### The unpacked PMI and CS are the inertia tensor and the cross-sections of the mesh, which can be got from Shipedit. To get it, click on Calc -> Start/continue MC integration. Wait until the numbers at ‘Parameters’ section are stable (Should be around 2,000,000 samples), then stop it from Calc -> Stop MC integration.

### 

### Then write the cross-sections as it (each number is separated with spaces), and write the marked inertia tensor number in order and separated with spaces (e.g. 0.17 3.17 3.14).

### Note that the unpacked inertia tensor and cross-sections are optional, and your cargo will work normally without them.

Cargo texturing

### Texturing a cargo is basically creating a new cargo from another one with a changed texture. You need to create a copy from that cargo configuration file, mesh, and texture.

### The cargo configuration file can be found in ‘Config\Vessels\UCSO’ folder. Make a copy and open it. Under ‘CargoMesh’ entry, you will find the cargo’s mesh filename.

### The mesh file can be found in ‘Meshes\UCSO’ folder. Its name is the value in ‘CargoMesh’ entry in the configuration file. Make a copy from it also. To know the cargo’s texture name, open it and scroll to the end. You’ll find something like this:

TEXTURES 1

UCSO\FlagEgypt.dds

### The texture file can be found in ‘Textures\UCSO’ folder. Its name is the name in the cargo’s mesh file. Make the texture you want, and rename it to a unique name to avoid conflicts with other add-ons (e.g. AR61CargoFuel.dds). The texture format is DDS.

### After finishing, rename your copy from the mesh and configuratio file, preferably the texture filename. The filename must be unique also. Open the mesh file and change the texture filename there to your texture filename, then open the configuration file and change the ‘CargoMesh’ to your mesh filename. Test your changes. If all looks great, publish your texture.

UCSO API

### To support UCSO in your vessel, you need to use the C++ API.

### First, you need to set your project. If not already done, you need to import Orbiter SDK properties. In your project .’vcxproj’ file, add the following lines:

<ImportGroup Condition="'$(Configuration)'=='Release'" Label="PropertySheets">

<Import Project="$(ProjectDir)’RELATIVE PATH’resources\Orbiter vessel.props" />

</ImportGroup>

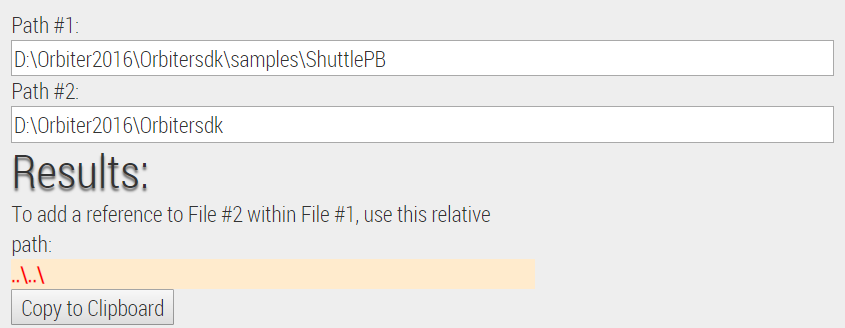
<ImportGroup Condition="'$(Configuration)'=='Debug'" Label="PropertySheets">

<Import Project="$(ProjectDir)’RELATIVE PATH’resources\Orbiter vessel.props" />

<Import Project="$(ProjectDir)’RELATIVE PATH’resources\Orbiter debug.props" />

</ImportGroup>

### You need to replace ‘RELATIVE PATH’ with the relative path from your project folder to ‘Orbitersdk’ folder. You can use [this calculator](https://www.stevebreese.com/Relative-Path-Calculator). Insert the project folder path in the first box, and the ‘Orbitersdk’ folder in the second box, then copy the first result and replace ‘RELATIVE PATH’ with it.



### The result is ..\..\, so the full path should be "$(ProjectDir)..\..\resources\Orbiter vessel.props". Do this for all projects if you have multiple projects in one solution.

### Use UCSO by including ‘UCSO.h’ file and linking against ‘UCSO.lib’ file. The API methods are explained in ‘UCSO.h’ file.

### The API doesn’t depend on UCSO, so it’ll work without it. The errors will be normal errors (e.g. no cargo in range while trying to grapple a cargo).

### The API depends on Orbiter API and STL.

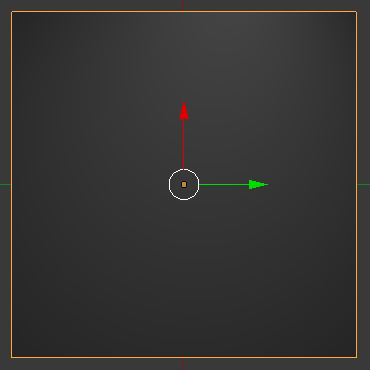
### You can know if UCSO is installed by calling GetUCSOVersion method, which will return 0 if UCSO isn’t installed.

### The standard keyboard shortcuts should be used unless your vessel is using them already. If so, this should be stated in your vessel manual.

### Otherwise, you can state that the standard UCSO shortcuts are used.

Attachment points

### The attachment point for UCSO cargo is at the center of the cargo bottom. You need to set your attachment point accordingly.



### The attachment point direction and rotation must be set properly, as the release velocity (if released in space) will be applied in the Y-axis.

### For example, if your attachment point is below the vessel, the direction and rotation must be set inverted to have the velocity applied below. Otherwise, the cargo will pass through the vessel.

### If your vessel has multiple slots, leave at least 1.5m between each cargo.

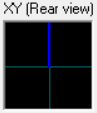
### A very handy tool is [Mesh Wizard](https://www.orbithangar.com/searchid.php?ID=2740), which will show you the attachment point axis. In order to run the program, you need to download [MSVBVM50.DLL](https://www.dll-files.com/msvbvm50.dll.html) and place it in the program folder, then open the command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

regsvr32 COMCTL32.OCX

regsvr32 MSFLXGRD.OCX

### Then open the program, it should open without problems. Open the ROT vector calculator from the ‘Calculator’ menu. For a normal attachment, the DIR is 0 0 1, and the ROT is 0 1 0.

### The XY or the rearview blue line represents the direction of release velocity. If it points up, the cargo will move up, and vice versa.



Credits

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

### [Woo482](https://www.orbiter-forum.com/member.php?u=195) for the ground release location code.

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

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

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

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

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

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

About

### UCSO is a free opensource cargo management system for Orbiter. It’s under GPL 3 license. The source code can be found on the [UCSO GitHub repository](https://github.com/abdullah-radwan/UCSO).

### All contribution are appreciated.

### Copyright © Abdullah Radwan