Welcome in Universal Cars & Cargos 3.0 SDK

This is the SDK to add cargo ability to vessels, for car or cargo designs see the html doc

What's new in version 3.0 & 2.5:

- Complete compatibility with Orbiter 2010 clients. (DX9, DX11, other)
- Unpack cargo bug cleared
- Release speed in space bug cleared (now smooth as a silk)
- New function “ScnEditor_SelectPreviousCargoAvailableOnDisk”

You just need to compile with the new UCGOCargoSDK.h and UCGOCargoSDK.lib without changing anything in your code.

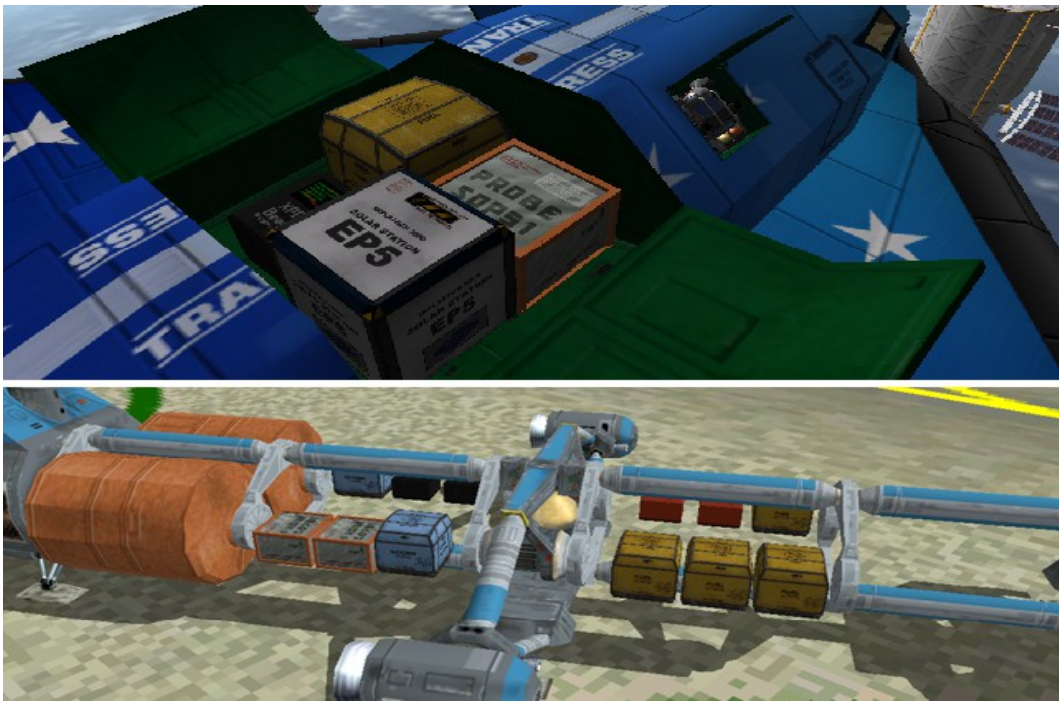
Content:

What is UCGO ?

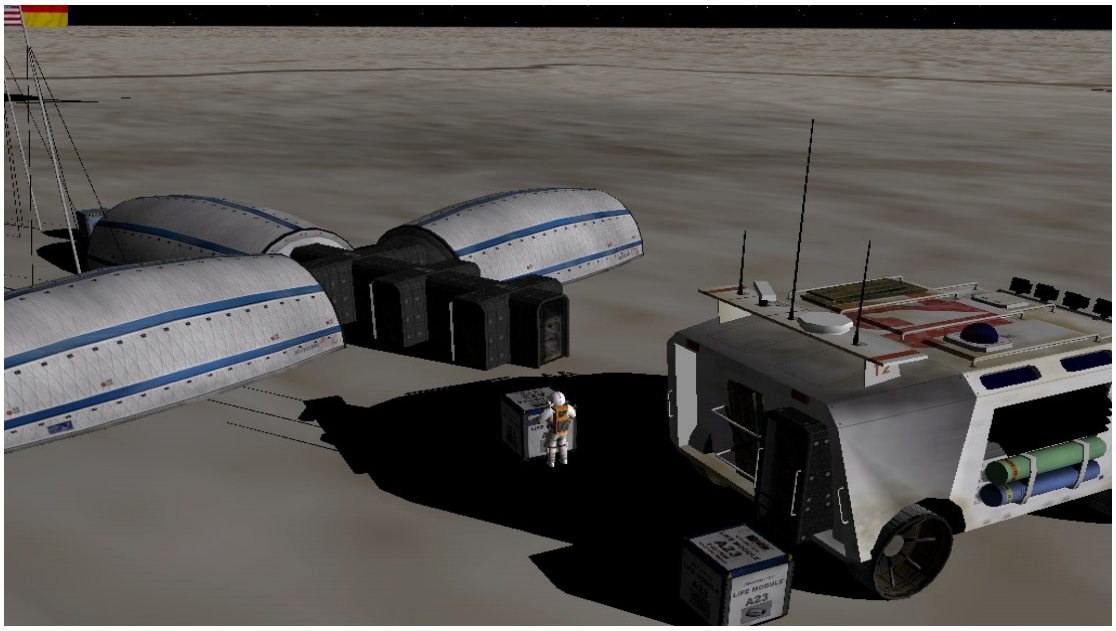
"UCGO" is the next logical step after UMmu in "universal" add-ons, allowing your vessels and UMmus to be more active in the Orbiter world. It's a set of SDKs to design cargo, cars, and allow you to add standard cargo support to any vessel.

One screen shot is worth a thousand words :)

UCGO allows you to add full cargo support to your ship with a few C++ lines. Up to 40 cargo slots are possible:



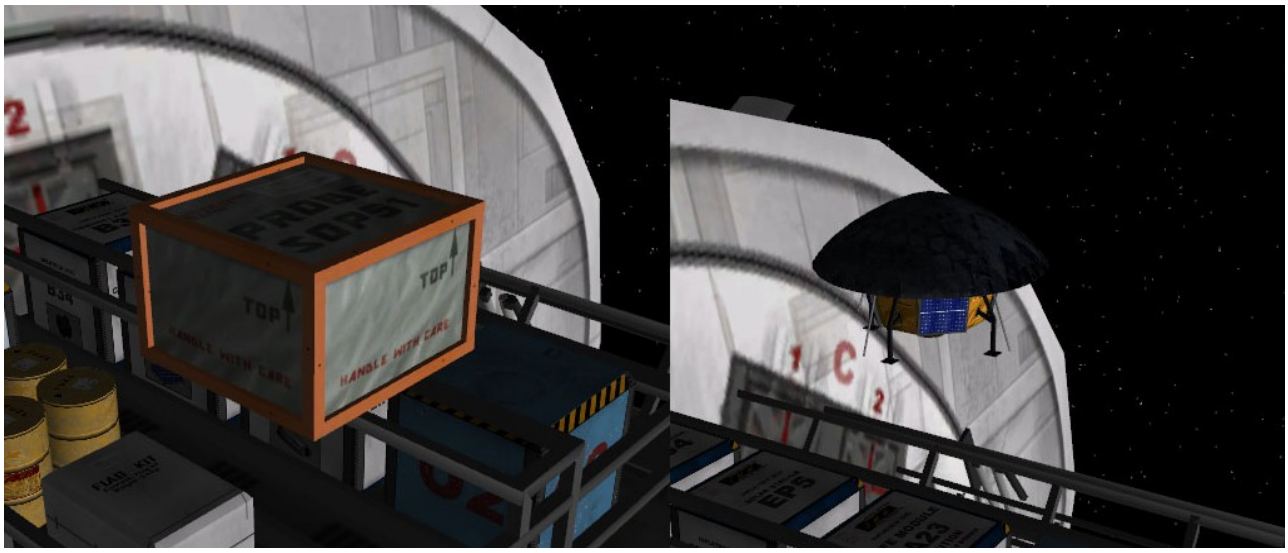
Here you can see cargo that expands into a base once unpacked by a UMmu. Anyone can add more cargo types.



Cars can carry UCGO cargo and even refuel any ships:

Cargo has chute support:

Cargo can even spawn active Vessels. Here is a cargo module that transform into an active probe that can deorbit itself. The possibilities are nearly endless:



Here is an example of cargo that expands to 5 beam lights once unpacked by UMmu:

Did I forgot to mention that your vessel would be able to "use" cargo contents? Fuel, Oxygen, Helium 3, gold or whatever you want. The SDK allows this and more!

New cars or cargo can be added by an author without one line of C++ programming. You only need a mesh and a config file. You can design new cargo, rovers, or cars to go with your add-on, but your add-on could also load and use all new cargo designed by other users.

"UCGO, the Next logical step !"
http://orbiter.dansteph.com/index.php?disp=d

License, UCGO redistributable files

LICENSE - IMPORTANT ! ! ! Please do not redistribute UCGO Files !

- The End User must also have the UCGO add-on installed. The add-on contains sound, .dll, meshes and various other files necessary for your add-on to function properly. BUT:

1-You are not allowed to redistribute the files of the UCGO basic installation with your add-on. Instead, you must inform the user by other means that your add-on requires UCGO, and that it must be downloaded from my site. This avoids a bad mix of versions. The installer checks the version before installation. This would not be the case with your add-on. I will continue to improve UCGO, and it would not be possible to do this if you redistributed outdated files.

2-You can, however, redistribute files made for this purpose in the SDK directory, drivers, meshes and such. If you really want to something distributed with UCGO please ask me first.

3-You can also distribute skinned meshes or textures of cars and cargos given you clearly state it's a skinned version with credit due to author. In this case howewer please do not overwrite one basic file of UCGO, copy the mesh, create a new config and add a NEW variant of one cargo or cars.

UCGO URL : : http://orbiter.dansteph.com

As with Ummu, I will maintain the compatibility across UCGO versions. A vessel that can carry UCGO 3.0 cargo, for example, will still work with future versions of UCGO. The same applies for cars and cargo.

"No puzzle add-on": If your add-on supports both UMmu and UCGO, you only have to give the link for UCGO It contains the UMmu 3.0 redistributable. See UMmu doc

Highly recommended to learn UCGO programming

The best way to learn is to see an example. Copy the whole folder "SDK/ShuttlePB_Example_UCGO" into the "sample" directory of the Orbiter SDK, and load the project into VC++. At compilation, it will create a "ShuttlePB.dll" in the release or debug directory that you must copy into the "modules" directory of Orbiter.

The documentation for each SDK function is in the header "UCGOCargoSDK.h". Please look here for more information on functions.

Note on cargo resource keywords

UCGO SDK functions allow vessels to consume cargo resources. The cargo designer can define the contents of the cargo with a keyword. Common examples would be to consume «fuel» or «oxygen» cargo to refill your vessel's tanks.

There is no limit on resource keyword. You can even define a cargo that contain «barbies» and use this keyword to refill your «barbies» tank. However, in order to define a common standard, I propose to all the cargo and vessel authors the following list of basic keywords:

oxygen, fuel, food, water, hydrogen, gold, helium, metal, rock, powercell

Using these keywords gives a greater chance that other authors have made cargo or a vessel that can use them too.

Note on cargo size

The size of the cargo must absolutely fit in a 1.3mx1.3mx1.3m max bounding box. Authors of cargo containers are not allowed to exceed this size. To check your cargo bay size you can use oxygen or fuel default cargo that have exactly this size (apart from the corners that are rounded).

Note on C++ compiler, link to free compiler and tutorial

Got the infamous LINK2001 error with your compiler?

Example :
error LNK2001: unresolved external symbol __ftol2_sse
error LNK2001: unresolved external symbol _sprintf_s

The library was compiled with VC++2005 (8.0 version), you should have your VC++ .lib up to date, otherwise you'll get errors. Fortunately there is a **FREE** version of Visual C++ Express that can compile your add-on with this new library. Upgrading to at least VC++ 2005 or express is a good thing anyway (we are in 2010 ;))! If your version is up to date and you have link errors, then it may be your compile settings. Again, see the «UCGO_ShutllePB_Example». If it compiles properly, compare the settings from your project with the example settings.

Please click link below to install Visual C++ express to compile Orbiter add-ons:
http://www.orbiterwiki.org/wiki/Free_Compiler_Setup

Have Fun and make great Orbiter Add-ons !

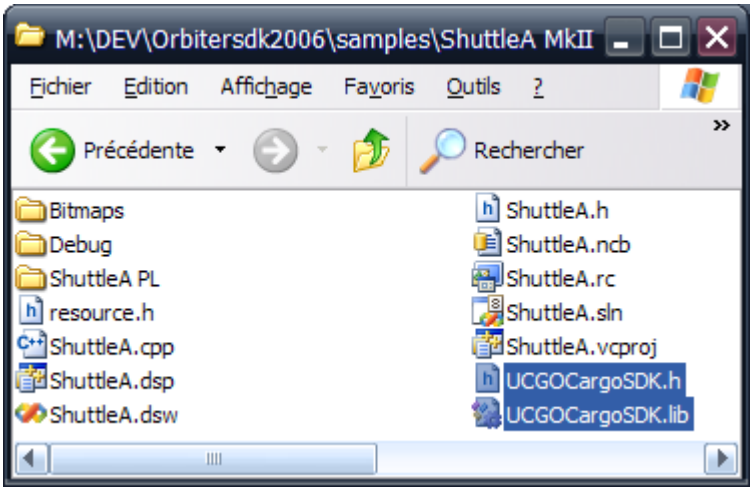
Daniel Polli (aka Dansteph)
http://orbiter.dansteph.com

2-Adding UCGO Cargo to Your Vessel, Complete Walk-through

This example uses a HUD message display and code that is ready to copy/paste for a fully functional UCGO cargo vessel. Once pasted and working, if you are an advanced user, it is very easy to modify for your own interface. See comments below:

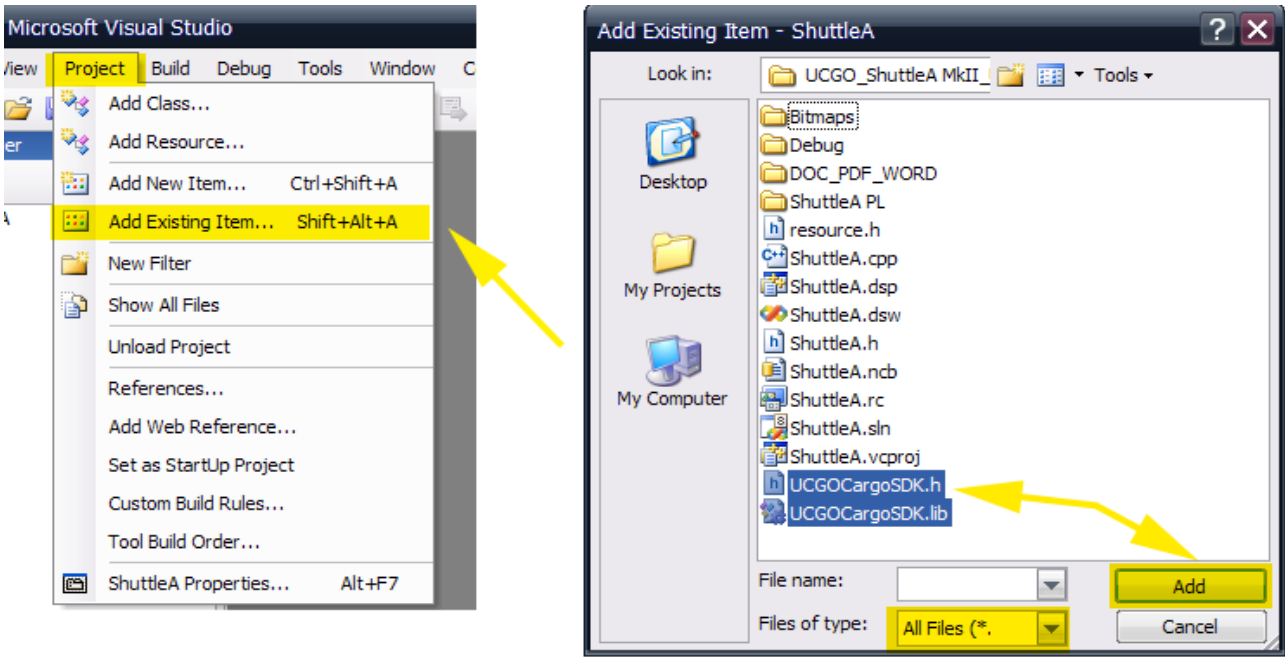
Step 1:

Copy « UCGOCargoSDK.h » and « UCGOCargoSDK.lib » located in « Doc/UniversalCarsAndCargos/SDK/UCGO_ShuttlePB_Example » to your project's folder:



Step 2:

Include them in your project:



Step 3:

Include the SDK header into your class header: (*.h file where your VESSEL class is declared)
Parts in Yellow below must be copied/pasted into your code at the correct location. Take great care to copy them exactly, one missing sign or wrong character and it will not work/compile.

```
// ShuttleA.h
// Class interface for Shuttle-A vessel class module
// ==============================================================

#ifndef __SHUTTLEA_H
#define __SHUTTLEA_H

#include "orbitersdk.h"
#include "UCGOCargoSDK.h" //copy past this line
```

Step 4:

In your Vessel's class declare the UCGO class handle functions and variables:

```
// ==============================================================
// Interface for derived vessel class: ShuttleA
// ==============================================================
class ShuttleA: public VESSEL2 {
public:
        ShuttleA (OBJHANDLE hObj, int fmodel);
        ~ShuttleA ();

        // UCGO 3.0 CLASS HANDLE FUNCTION AND VARIABLES
        UCGO    hUcgo;                          // Cargo class handle
        char   *SendCargHudMessage(void);   // Cargo hud display function
        char    cCargoHudDisplay[255];          // Cargo hud display char variable
        double  dCargHudMessageDelay;           // Cargo hud display delay
        int     iSelectedCargo;                 // for the selection of cargos -1 by default
```

Step 5:

Note: For a complete working example see « ShuttlePB_Example_UCGO » project. Copy this project to OrbiterSDK/sample and open it in visual C++. You can also copy code from this example instead of this document. The code is the same but it might be more practical.

Now in clbkSetClassCap include the essential initialization. Of course you should call the functions with your own parameters (see **UCGOCargoSDK.h** and annexes for functions parameter descriptions)

NOTE: You can add up to 40 slots. Cargo have a normalized size of a maximum of 1.3mx1.3mx1.3m (1.3 meters cubed). Make sure to make enough room in your bay. It is a good idea to leave a 10cm space between slots. (Here below the second slot is at Z position 3+1.4meter from slot one=4.4m.)

```
// --------------------------------------------------------------
// Set vessel class caps
// --------------------------------------------------------------
void ShuttleA::clbkSetClassCaps (FILEHANDLE cfg)
{
        //UCGO 3.0 Initialisation, cargo slot pos, rot declaration
        hUcgo.Init(GetHandle());
        hUcgo.DeclareCargoSlot(0,_V(0,2.0,3.0),_V(0,0,0)); // slot 0
        hUcgo.DeclareCargoSlot(1,_V(0,2.0,4.4),_V(0,0,0)); // slot 1
```

Step 6:

Still in clbkSetClassCap but after all other declarations, we'll now set some general parameters for UCGO: Maximum cargo mass that your vessel will accept, release position on ground when landed, grapple distance, release speed in space etc.
NOTE: You can define a release position on the ground per slot, again see the SDK's header for complete list of functions and parameters.

```
//COPY ONLY PARTS IN YELLOW, THIS IS JUST TO SHOW WHERE IT MUST BE COPIED
hUcgo.Init(GetHandle());
hUcgo.DeclareCargoSlot(0,_V(0,2.0,3.0),_V(0,0,0)); // slot 0
hUcgo.DeclareCargoSlot(1,_V(0,2.0,4.4),_V(0,0,0)); // slot 1

// UCGO 3.0 Parameters settings
hUcgo.SetReleaseSpeedInSpace(0.10f);          // release speed of cargo in space in m/s
hUcgo.SetMaxCargoMassAcceptable(5000.0);  // max cargo mass in kg that your vessel can carry
hUcgo.SetGrappleDistance(50);                    // grapple distance radius in meter from center of ship
hUcgo.SetGlobalGroundReleasePos(_V(0,0,4));// global release position on ground
```

Step 7:

Just after the initialization above, add the variables initialization required for this demo.

```
//COPY ONLY PART IN YELLOW, THIS IS JUST TO SHOW WHERE IT MUST BE COPIED
hUcgo.SetGrappleDistance(50);                    // grapple distance radius in meter from center of ship
hUcgo.SetGlobalGroundReleasePos(_V(0,0,4));// global release position on ground

// UCGO Variables initialisation
cCargoHudDisplay[0]=0;                    // Cargo hud display char variable
dCargHudMessageDelay=0;      // Cargo hud display delay
iSelectedCargo=-1;               // for the selection of cargos (-1 mean "default" see header)
// welcome message with keys for users
strcpy(SendCargHudMessage(),"Cargo key: C/SHF+C = grapple/release, 9/SHF+9 = add cargo from disk, 8=infos on cargos");
```

Step 7.5: OPTIONAL Virtual cargo slot doors

By default, this demo doesn't show cargo bay door management. Those «virtual doors» would forbid any grappling/release if closed. You can manage them globally or per slot. If you have a cargo door animation, this is the function to use:

NOTE: by default all virtual doors are open

```
// Example set global doors state for ALL slots
hUcgo.SetSlotDoorState(TRUE);              // TRUE=open FALSE=closed

// Example set doors state for One slots in this example the slot Nbr 0
hUcgo.SetSlotDoorState(TRUE,0);          // TRUE=open FALSE=closed
```

Step 8:

Add the automatic loading/saving of UCGO cargo in clbkLoadStateEx and clbkSaveState:

```
// --------------------------------------------------------------
// Write status to scenario file
// --------------------------------------------------------------
void ShuttleA::clbkSaveState (FILEHANDLE scn)
{
	char cbuf[256];
	// default vessel parameters
	VESSEL2::clbkSaveState (scn);

	// Save UCGO 3.0 cargo in scenario
	hUcgo.SaveCargoToScenario(scn);
```

```
// --------------------------------------------------------------
// Read status from scenario file
// --------------------------------------------------------------
void ShuttleA::clbkLoadStateEx (FILEHANDLE scn, void *vs)
{
	char *line;
	while (oapiReadScenario_nextline (scn, line))
	{
		if(hUcgo.LoadCargoFromScenario(line)==TRUE) // UCGO load cargo
			continue;
```

Step 9:

Copy the member function that will fill our char value with cargo message and set HUD display delay in seconds. If you don't know where to copy it, follow the example below and paste it near clbkSetClasCap . Don't forget to change the class' name for the class name of your vessel (in red)

NOTE: if you want to customize UCGO in your vessel «cCargoHudDisplay» is a 255 character value that will always contain the last cargo operation message.

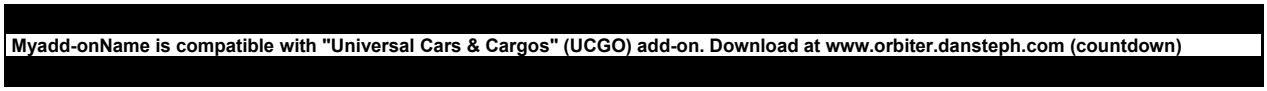
```
//////////////////////////////////////////////////////
// SendCargHudMessage
//////////////////////////////////////////////////////
char *ShuttleA::SendCargHudMessage(void)
{
	dCargHudMessageDelay=15; // 15 seconds display delay for msg
	return cCargoHudDisplay;
}
// --------------------------------------------------------------
// Set vessel class caps
// --------------------------------------------------------------
NAME IN RED ABOVE MUST BE THE SAME THAN THIS ONE BELOW-->
void ShuttleA::clbkSetClassCaps (FILEHANDLE cfg)
```

Step 10:

This is an important function! It will warn users during the first 20 seconds of the scenario if they not have UCGO 3.0 installed, or if the installation is outdated (in those cases the user would miss all the awesome cargo features of your vessel).
I recommend placing it at the end of clbkPostStep or clbkPreStep so the message, which uses oapiDebugString, cannot be overwritten by another debug message. You must pass your add-on's name as the parameter.

```
// --------------------------------------------------------------
// Frame update
// --------------------------------------------------------------
void ShuttleA::clbkPostStep (double simt, double simdt, double mjd)
{
        // At the very end of clbkPostStep or clbkPreStep
        hUcgo.WarnUserUCGONotInstalled("Myadd-onName");
}
```

Warning text duration is 20 seconds, and the text is displayed in the bottom of Orbiter screen as follows:



NOTE: For our performance lovers, you could add this line above 100,000 times before seeing any impact on frame rates, so ;)

NOTE: If you also have UMMU 3.0 in your vessel, place this line AFTER the UMMU warning line. Recall that the UCGO installation adds UMMU 3.0, so you only need to ask the user to install the UCGO add-on if your vessel is UMMU & UCGO compatible.

UMMU support is highly recommended if you have UCGO. See Doc/UMMU_SDK for a complete copy/paste walk-through.

Step 11:

Now in clbkDrawHud add the following:

```
// --------------------------------------------------------------
// Setup the virtual cockpit instruments
// --------------------------------------------------------------
void ShuttleA::clbkDrawHUD (int mode, const HUDPAINTSPEC *hps, HDC hDC)
{
        // draw the default HUD
        VESSEL2::clbkDrawHUD (mode, hps, hDC);

        // UCGO display messages
        if(dCargHudMessageDelay>0)
        {
                TextOut (hDC,5,hps->H/60*12,cCargoHudDisplay,strlen(cCargoHudDisplay));
                dCargHudMessageDelay-=oapiGetSimStep();
                if(dCargHudMessageDelay<0)
                        dCargHudMessageDelay=0;
        }
```

Step 12:

Now in clbkVisualCreated add the following:

```
// --------------------------------------------------------------
// clbkVisualCreated
// --------------------------------------------------------------
void ShuttleA::clbkVisualCreated (VISHANDLE vis, int refcount)
{
	hUcgo.SetUcgoVisual(vis);		// must be called in clbkVisualCreated.
}
```

Step 13:

Now you must allow users to control the Cargo. I show keyboard controls here, but you can use your own methods: (i.e. panel or VC buttons).

Key «C» grapples a cargo container, key «SHIFT+C» releases a cargo container, key «9» cycles and selects one cargo container on your hard drive (without pulling it from inventory using the scenario editor ;)), key SHIFT+9 adds the last selected cargo container, key «9» and key «8» show some information on loaded cargo. This should be included in clbkConsumeBufferedKey.

```
// --------------------------------------------------------------
// Respond to buffered keyboard events
// --------------------------------------------------------------
int ShuttleA::clbkConsumeBufferedKey (DWORD key, bool down, char *kstate)
{
	if (!down) return 0;	// only process keydown events

	// 9 key "select" one cargo on disk (cycle)
	if(key==OAPI_KEY_9&&!KEYMOD_SHIFT(kstate)&&!KEYMOD_CONTROL (kstate))
	{
		sprintf(SendCargHudMessage(),"%s - selected",hUcgo.ScnEditor_SelectNextCargoAvailableOnDisk());
		return 1;
	}
	// SHIFT+9 key "add last cargo selected by key 9"
	// If iSelectedCargo=-1 (default) add to the first free slot found
	if(key==OAPI_KEY_9&&KEYMOD_SHIFT(kstate)&&!KEYMOD_CONTROL (kstate))
	{
		if(hUcgo.ScnEditor_AddLastSelectedCargoToSlot(iSelectedCargo)==TRUE)
		{
			strcpy(SendCargHudMessage(),"Cargo added to ship");
		}
		else
		{
			if(iSelectedCargo<0)
			{
				strcpy(SendCargHudMessage(),"Cargo not added (ship full or weight excess ?)");
			}
			else
			{
				strcpy(SendCargHudMessage(),"Cargo not added (slot full ship full or weight excess ?)");
			}
		}
		return 1;
	}
	// "C" grapple cargo. If iSelectedCargo=-1 (default) add to the first free slot found
	if(key==OAPI_KEY_C&&!KEYMOD_SHIFT(kstate)&&!KEYMOD_CONTROL (kstate))
	{
		int ReturnedCode=hUcgo.GrappleOneCargo(iSelectedCargo);
		// for return code list see function "GrappleOneCargo" in the header
		switch(ReturnedCode)
```

```
		{
		case 1:
			strcpy(SendCargHudMessage(),"Cargo grappled");
			break;
		case 0:
			strcpy(SendCargHudMessage(),"No cargo in range");
			break;
		case -1:
			strcpy(SendCargHudMessage(),"cargo exceed maximum mass");
			break;
		case -2:
			strcpy(SendCargHudMessage(),"bad config, mesh not found or slot not declared");
			break;
		case -3:
			strcpy(SendCargHudMessage(),"Can't grapple cargo, slot not empty");
			break;
		case -4:
			strcpy(SendCargHudMessage(),"Can't grapple cargo,doors closed ");
			break;
		case -5:
			strcpy(SendCargHudMessage(),"Can't grapple cargo,Ship full");
			break;
		default:
			strcpy(SendCargHudMessage(),"Misc error Unable to grapple cargo");
		}
		return 1;
	}
	// SHIFT+C release cargo. If iSelectedCargo=-1 (default) release the first free slot found
	if(key==OAPI_KEY_C&&KEYMOD_SHIFT(kstate)&&!KEYMOD_CONTROL (kstate))
	{
		if(hUcgo.ReleaseOneCargo(iSelectedCargo)!=FALSE)
		{
			strcpy(SendCargHudMessage(),"Cargo released");
		}
		else
		{
			strcpy(SendCargHudMessage(),"Cargo not released (empty slot, no cargo aboard?)");
		}
		return 1;
	}
	// "8" show some info on cargo
	if(key==OAPI_KEY_8&&!KEYMOD_SHIFT(kstate)&&!KEYMOD_CONTROL (kstate))
	{
		sprintf(SendCargHudMessage(),"%i cargos aboard. "
		"Total cargos weight: %.0fkg",hUcgo.GetNbrCargoLoaded(),hUcgo.GetCargoTotalMass());
		return 1;
	}
```

Congratulations! Your vessel is now fully UCGO 3.0 compatible.
But, you can do a lot more using SDK functions. See SDK header.

Annex A – Useful Snippets of Code

Refill tanks with fuel cargo using SDK functions

The SDK includes a function to “consume” rcargo resources aboard or in the vicinity (grappling distance) of the vessel. Resources are defined by keywords in cargo's config file (see above about keywords).

The cargo mass will be decreased by the amount of resources consumed, and Orbiter will remember the decrease in mass. If all resources of one cargo container is used, it disappears. You can use cargo systems to do common things, but you can also use it to make cool new things! For example, your engine could consume “nuclear” cargo specifically designed for your vessel. You can also require the use of power-cell cargo that would supply your ship with electricity. Ideas are endless.

See the **UCGOCargoSDK.h** header to learn more about the consume function available.

Here is a simple example of a code that would consume fuel cargo to refill your fuel tank:
It uses the HUD feedback function, but of course you can do you own.

```
double dNeededFuel=MAX_MAIN_FUEL-GetPropellantMass (ph_main);
if(dNeededFuel<20)
{
        strcpy(SendCargHudMessage(),"Fuel tanks almost full");
        return 1;
}
double dEatenFuel=hUcgo.EatCloserFuelCargo(dNeededFuel);
if(dEatenFuel<0)
{
        strcpy(SendCargHudMessage(),"no fuel cargo found aboard or in vincinity");
        return 1;
}
// refill
SetPropellantMass (ph_main,GetPropellantMass (ph_main)+dEatenFuel);
sprintf(SendCargHudMessage(),"%.2fkg of fuel added to tank",dEatenFuel);
```

BAD IDEA:
Do not «consume» resources of cargo each frame ! In the case of the power-cell electricity idea above, for example, use an internal custom tank (a variable) that you refill from time to time with cargo resources.