UCSO Vessel API Walkthrough

###### In this walkthrough, we will add UCSO support to the default ShuttlePB. You must set up the project as detailed in the manual. You can find the completed example in ‘Orbitersdk/samples/UCSO/ShuttlePB\_UCSO’.

Start by including the vessel API header file and making an instance of it. Add clbkConsumeBufferedKey to intercept keyboard commands, clbkDrawHUD to draw on the HUD, and clbkPreStep to set the messages timer.

#include <UCSO\_Vessel.h>

………………………

public:

………………………

int clbkConsumeBufferedKey(DWORD key, bool down, char\* kstate);

bool clbkDrawHUD(int mode, const HUDPAINTSPEC\* hps, oapi::Sketchpad\* skp);

void clbkPreStep(double simt, double simdt, double mjd);

private:

………………………

UCSO::Vessel\* ucso;

ATTACHMENTHANDLE attachmentHandle = nullptr;

int cargoIndex = 0; // For the cargo selection on the HUD

char buffer[256]; // To draw on the HUD

const char\* message; // To show messages on the HUD

double timer = 0; // To show the messages for 5 seconds on the HUD

};

Initialize the API instance in the constructor and destroy it in the destructor.

ShuttlePB::ShuttlePB(OBJHANDLE hVessel, int flightmodel) : VESSEL3(hVessel, flightmodel)

{

ucso = UCSO::Vessel::CreateInstance(this);

sprintf(buffer, "UCSO version: %s", ucso->GetUCSOVersion());

message = \_strdup(buffer);

}

ShuttlePB::~ShuttlePB() { delete ucso; }

Add the cargo slot and attachment handle to the API in the clbkSetClassCaps method.

void ShuttlePB::clbkSetClassCaps (FILEHANDLE cfg)

{

………………………

attachmentHandle = CreateAttachment(false, { 0,-1,-1.3 }, { 0,-1,0 }, { 0,0,1 }, "UCSO");

ucso->SetSlotAttachment(0, attachmentHandle);

}

Intercept the keyboard commands to redirect them to the API and set the message to show the result.

int ShuttlePB::clbkConsumeBufferedKey(DWORD key, bool down, char\* kstate)

{

if (!down) return 0;

if (KEYMOD\_SHIFT(kstate)) // If Shift key is pressed

{

switch (key)

{

case OAPI\_KEY\_A:

switch (ucso->AddCargo(cargoIndex))

{

case UCSO::Vessel::GRAPPLE\_SLOT\_OCCUPIED:

message = "Couldn't add the selected cargo: the slot is occupied.";

break;

case UCSO::Vessel::GRAPPLE\_SUCCEEDED:

message = "The selected cargo is added successfully.";

break;

case UCSO::Vessel::NO\_CARGO\_IN\_RANGE:

message = "Couldn't add the selected cargo: the index is invalid.";

break;

case UCSO::Vessel::GRAPPLE\_FAILED:

message = "Couldn't add the selected cargo.";

break;

default: break;

}

timer = 0;

return 1;

case OAPI\_KEY\_G:

switch (ucso->GrappleCargo())

{

case UCSO::Vessel::GRAPPLE\_SUCCEEDED:

message = "The nearest cargo is grappled successfully.";

break;

case UCSO::Vessel::NO\_CARGO\_IN\_RANGE:

message = "Couldn't grapple cargo: no grappleable cargo in range.";

break;

case UCSO::Vessel::GRAPPLE\_SLOT\_OCCUPIED:

message = "Couldn't grapple cargo: the slot is occupied.";

break;

case UCSO::Vessel::GRAPPLE\_FAILED:

message = "Couldn't grapple cargo";

break;

default: break;

}

timer = 0;

return 1;

case OAPI\_KEY\_R:

switch (ucso->ReleaseCargo())

{

case UCSO::Vessel::RELEASE\_SUCCEEDED:

message = "The grappled cargo is released successfully.";

break;

case UCSO::Vessel::NO\_EMPTY\_POSITION:

message = "Couldn't release the grappled cargo: no empty position nearby.";

break;

case UCSO::Vessel::RELEASE\_SLOT\_EMPTY:

message = "Couldn't release the grappled cargo: the slot is empty.";

break;

case UCSO::Vessel::RELEASE\_FAILED:

message = "Couldn't release the grappled cargo.";

break;

default: break;

}

timer = 0;

return 1;

case OAPI\_KEY\_P:

if (ucso->PackCargo()) message = "The nearest cargo is packed successfully.";

else message = "Couldn't pack the nearest cargo: no packable cargo in range or the packing failed.";

timer = 0;

return 1;

case OAPI\_KEY\_U:

if (ucso->UnpackCargo()) message = "The nearest cargo is unpacked successfully.";

else message = "Couldn't unpack the nearest cargo: no unpackable cargo in range or the unpacking failed.";

timer = 0;

return 1;

case OAPI\_KEY\_F:

{

double requiredMass = GetMaxFuelMass() - GetFuelMass();

// Drain the required mass to fill the tank, by subtracting the maximum mass from the current mass

double drainedMass = ucso->DrainCargoResource("fuel", requiredMass);

// If no resource cargo is available, drain from the nearest station or unpacked resource

if(drainedMass == 0) drainedMass = ucso->DrainStationOrUnpackedResource("fuel", requiredMass);

if (drainedMass > 0)

{

SetFuelMass(GetFuelMass() + drainedMass);

sprintf(buffer, "%g kilograms of fuel was drained", drainedMass);

message = \_strdup(buffer);

}

else message = "Couldn't drain fuel.";

timer = 0;

return 1;

}

case OAPI\_KEY\_D:

switch (ucso->DeleteCargo(0))

{

case UCSO::Vessel::RELEASE\_SUCCEEDED:

message = "The grappled cargo is deleted successfully.";

break;

case UCSO::Vessel::RELEASE\_SLOT\_EMPTY:

message = "Couldn't delete the grappled cargo: the slot is empty.";

break;

case UCSO::Vessel::RELEASE\_FAILED:

message = "Couldn't delete the grappled cargo.";

break;

default: break;

}

timer = 0;

return 1;

default: break;

}

}

if (key == OAPI\_KEY\_S)

{

// Reset the index if reached the cargo count, otherwise increase the index

cargoIndex + 1 < ucso->GetAvailableCargoCount() ? cargoIndex++ : cargoIndex = 0;

return 1;

}

return 0;

}

Display the attached cargo information and the operation results on the HUD.

bool ShuttlePB::clbkDrawHUD(int mode, const HUDPAINTSPEC\* hps, oapi::Sketchpad\* skp)

{

// Draw the default HUD (Surface, Orbit, etc...)

VESSEL4::clbkDrawHUD(mode, hps, skp);

// Determine the screen ratio

int s = hps->H;

double d = s \* 0.00130208;

int sw = hps->W;

int lw = static\_cast<int>(16 \* sw / 1024);

int x = 0;

if (s / sw < 0.7284) x = (lw \* 10) + 10;

int y = static\_cast<int>((168 \* d) + (-88 \* d));

// Set the color to green

skp->SetTextColor(0x0066FF66);

sprintf(buffer, "Selected cargo to add: %s", ucso->GetAvailableCargoName(cargoIndex));

skp->Text(x, y, buffer, strlen(buffer));

y += 36;

skp->Text(x, y, "S = Select a cargo to add", 25);

y += 20;

skp->Text(x, y, "Shift + A = Add the selected cargo", 34);

y += 20;

skp->Text(x, y, "Shift + G = Grapple the nearest cargo", 37);

y += 20;

skp->Text(x, y, "Shift + R = Release the grappled cargo", 38);

y += 20;

skp->Text(x, y, "Shift + P = Pack the nearest cargo", 34);

y += 20;

skp->Text(x, y, "Shift + U = Unpack the nearest cargo", 36);

y += 20;

skp->Text(x, y, "Shift + F = Drain the nearest fuel resource", 43);

y += 20;

skp->Text(x, y, "Shift + D = Delete the grappled cargo", 37);

// Display the message if the timer is below 5

if (timer < 5) { y += 36; skp->Text(x, y, message, strlen(message)); }

UCSO::Vessel::CargoInfo cargoInfo = ucso->GetCargoInfo(0);

if (!cargoInfo.valid) return true;

y += 36;

skp->Text(x, y, "Grappled cargo information", 26);

y += 40;

sprintf(buffer, "Name: %s", cargoInfo.name);

skp->Text(x, y, buffer, strlen(buffer));

y += 20;

sprintf(buffer, "Mass: %gkg", cargoInfo.mass);

skp->Text(x, y, buffer, strlen(buffer));

switch (cargoInfo.type)

{

case UCSO::Vessel::STATIC:

skp->Text(x, y - 40, "Type: Static", 12);

break;

case UCSO::Vessel::RESOURCE:

skp->Text(x, y - 40, "Type: Resource", 14);

y += 20;

sprintf(buffer, "Resource : %s", cargoInfo.resource);

skp->Text(x, y, buffer, strlen(buffer));

y += 20;

sprintf(buffer, "Resource mass: %gkg", cargoInfo.resourceMass);

skp->Text(x, y, buffer, strlen(buffer));

break;

case UCSO::Vessel::UNPACKABLE\_ONLY:

skp->Text(x, y - 40, "Type: Unpackable only", 21);

y += 20;

sprintf(buffer, "Unpacked spawn count: %d cargo(es)", cargoInfo.spawnCount);

skp->Text(x, y, buffer, strlen(buffer));

case UCSO::Vessel::PACKABLE\_UNPACKABLE:

if (cargoInfo.type == UCSO::Vessel::PACKABLE\_UNPACKABLE) skp->Text(x, y - 40, "Type: Packable and unpackable", 29);

y += 20;

switch (cargoInfo.unpackingType)

{

case UCSO::Vessel::UCSO\_RESOURCE:

skp->Text(x, y, "Unpacking type: UCSO Resource", 29);

y += 20;

break;

case UCSO::Vessel::UCSO\_MODULE:

skp->Text(x, y, "Unpacking type: UCSO Module", 27);

y += 20;

sprintf(buffer, "Breathable: %s", cargoInfo.breathable ? "Yes" : "No");

skp->Text(x, y, buffer, strlen(buffer));

break;

Set the timer in the clbkPreStep method.

void ShuttlePB::clbkPreStep(double simt, double simdt, double mjd) { if (timer < 5) timer += simdt; }

case UCSO::Vessel::ORBITER\_VESSEL:

skp->Text(x, y, "Unpacking type: Orbiter vessel", 30);

y += 20;

sprintf(buffer, "Spawn module: %s", cargoInfo.spawnModule);

skp->Text(x, y, buffer, strlen(buffer));

y += 20;

switch (cargoInfo.unpackingMode)

{

case UCSO::Vessel::LANDING:

skp->Text(x, y, "Unpacking mode: Landing", 23);

break;

case UCSO::Vessel::DELAYING:

skp->Text(x, y, "Unpacking mode: Delaying", 24);

y += 20;

sprintf(buffer, "Unpacking delay: %is", cargoInfo.unpackingDelay);

skp->Text(x, y, buffer, strlen(buffer));

break;

case UCSO::Vessel::MANUAL:

skp->Text(x, y, "Unpacking mode: Manual", 22);

break;

}

break;

case UCSO::Vessel::CUSTOM\_CARGO:

skp->Text(x, y, "Unpacking type: Custom cargo", 28);

y += 20;

sprintf(buffer, "Breathable: %s", cargoInfo.breathable ? "Yes" : "No");

skp->Text(x, y, buffer, strlen(buffer));

break;

}

break;

}

return true;

}

Compile the project, spawn your vessel in a scenario using the scenario editor, and spawn some cargoes. Have fun!