```
// Simple airplane script example
  // Retrieved from Free SL Scripts on www.gendersquare.org/sl
// THIS SCRIPT IS PUBLIC DOMAIN! Do not delete the credits at the top of this script!
// Nov 25, 2003 - created by Andrew Linden and posted in the Second Life scripting forum// Jan
05, 2004 - Cubey Terra - minor changes: customized controls, added enable/disable physics
events
// Feel free to copy, modify, and use this script.
// Always give credit to Andrew Linden and all people who modify it in a readme or in the object
description.
// assumes that the root primitive is oriented such that its
// local x-axis points toward the nose of the plane, and its// local z-axis points toward the top
// control flags that we set laterinteger gAngularControls = 0;integer gLinearControls = 0;
// we keep track of angular history for more responsive turnsinteger gOldAngularLevel = 0;
// the linear motor uses an accumulator model rather than keeping track// of the linear control
level history                                                                          vector
gLinearMotor = <0, 0, 0>;
default{    state_entry()    {        llSetSitText("Fly");        llCollisionSound("", 0.0);
        // the sit and camera placement is very shape dependent        // so modify these to suit
your vehicle                                                                      llSitTarget(<0.6,
0.0, 0.20>, ZERO_ROTATION);
        llSetCameraEyeOffset(<-10.0, 0.0, 2.0> );
        llSetCameraAtOffset(<3.0, 0.0, 1.0> );
        llSetVehicleType(VEHICLE_TYPE_AIRPLANE);
        // weak angular deflection
llSetVehicleFloatParam(VEHICLE_ANGULAR_DEFLECTION_EFFICIENCY, 0.1);
        llSetVehicleFloatParam(VEHICLE_ANGULAR_DEFLECTION_TIMESCALE, 1.0);
        // strong linear deflection
llSetVehicleFloatParam(VEHICLE_LINEAR_DEFLECTION_EFFICIENCY, 1.0);
        llSetVehicleFloatParam(VEHICLE_LINEAR_DEFLECTION_TIMESCALE, 0.2);
        // somewhat responsive linear motor
llSetVehicleFloatParam(VEHICLE_LINEAR_MOTOR_TIMESCALE, 0.5);
        llSetVehicleFloatParam(VEHICLE_LINEAR_MOTOR_DECAY_TIMESCALE, 20);
        // somewhat responsive angular motor, but with 3 second decay timescale
llSetVehicleFloatParam(VEHICLE_ANGULAR_MOTOR_TIMESCALE, 0.5);
        llSetVehicleFloatParam(VEHICLE_ANGULAR_MOTOR_DECAY_TIMESCALE, 3);
        // very weak friction
//llSetVehicleVectorParam(VEHICLE_LINEAR_FRICTION_TIMESCALE, <1000.0, 1000.0,
1000.0> ); // CUBEY - original line                                llSetVehicleVectorParam(
VEHICLE_LINEAR_FRICTION_TIMESCALE, <200, 20, 20> ); // CUBEY - increased friction
        llSetVehicleVectorParam(VEHICLE_ANGULAR_FRICTION_TIMESCALE, <1000.0,
1000.0, 1000.0> );
        llSetVehicleFloatParam(VEHICLE_VERTICAL_ATTRACTION_EFFICIENCY, 0.65);  //
almost wobbly - CUBEY - increased from .25 to improve stability
llSetVehicleFloatParam(VEHICLE_VERTICAL_ATTRACTION_TIMESCALE, 1.5);    // mediocre
response
        llSetVehicleFloatParam(VEHICLE_BANKING_EFFICIENCY, 0.4);    // medium strength
```

```
   llSetVehicleFloatParam(VEHICLE_BANKING_TIMESCALE, 0.1);     // very responsive
       llSetVehicleFloatParam(VEHICLE_BANKING_MIX, 0.95);         // more banking when
moving
       // hover can be better than sliding along the ground during takeoff and landing        // but it
only works over the terrain (not objects)
       //llSetVehicleFloatParam(VEHICLE_HOVER_HEIGHT, 3.0);
       //llSetVehicleFloatParam(VEHICLE_HOVER_EFFICIENCY, 0.5);
       //llSetVehicleFloatParam(VEHICLE_HOVER_TIMESCALE, 2.0);
       //llSetVehicleFlags(VEHICLE_FLAG_HOVER_UP_ONLY);
       // non-zero buoyancy helps the airplane stay up        // set to zero if you don't want this
crutch
 llSetVehicleFloatParam(VEHICLE_BUOYANCY, 0.2);
       // define these here for convenience later         // CUBEY - modified these as per Shadow's
prefs                                                          gAngularControls =
CONTROL_RIGHT | CONTROL_LEFT | CONTROL_ROT_RIGHT | CONTROL_ROT_LEFT |
CONTROL_BACK | CONTROL_FWD;
 gLinearControls = CONTROL_UP | CONTROL_DOWN;
       llSetStatus(STATUS_PHYSICS, FALSE); //CUBEY - ensure that physics are disabled
when plane is rezzed so it doesn't fly off    }
    changed(integer change)    {       if (change & CHANGED_LINK)       {           key agent =
llAvatarOnSitTarget();
          if (agent)
          {
             if (agent != llGetOwner())
             {
                // only the owner can use this vehicle
                llSay(0, "You aren't the owner -- only the owner can fly this plane.");
                llUnSit(agent);
                llPushObject(agent, <0,0,10>, ZERO_VECTOR, FALSE);
             }
             else
             {
                // clear linear motor on successful sit
                gLinearMotor = <0, 0, 0>;
                 llSetVehicleVectorParam(VEHICLE_LINEAR_MOTOR_DIRECTION,
gLinearMotor);
                //llSetStatus(STATUS_PHYSICS, TRUE);
llSetVehicleFloatParam(VEHICLE_LINEAR_FRICTION_TIMESCALE, 1000.0);
                 llSetVehicleFloatParam(VEHICLE_ANGULAR_FRICTION_TIMESCALE, 1000.0);
                llRequestPermissions(agent, PERMISSION_TRIGGER_ANIMATION |
PERMISSION_TAKE_CONTROLS);
             }
          }
          else
          {
             // stop the motors
```

```
                gLinearMotor = <0, 0, 0>;
                 llSetVehicleVectorParam(VEHICLE_LINEAR_MOTOR_DIRECTION, gLinearMotor);
                 llSetVehicleVectorParam(VEHICLE_ANGULAR_MOTOR_DIRECTION,
gLinearMotor);
                // use friction to stop the vehicle rather than pinning it in place
 //llSetStatus(STATUS_PHYSICS, FALSE);
                 llSetVehicleFloatParam(VEHICLE_LINEAR_FRICTION_TIMESCALE, 1.0);
                 llSetVehicleFloatParam(VEHICLE_ANGULAR_FRICTION_TIMESCALE, 1.0);
                // driver is getting up              llReleaseControls();              llStopAnimation("sit");
           llSetStatus(STATUS_PHYSICS, FALSE); //CUBEY - turn off physics to make sure the
parked plane can't be moved
            }
        }
    }
    run_time_permissions(integer perm)    {        if (perm)        {            llStartAnimation("sit");
     llTakeControls(gAngularControls | gLinearControls, TRUE, FALSE);
            llSetStatus(STATUS_PHYSICS, TRUE); //CUBEY - enable physics when avatar sits
        }
    }
    control(key id, integer level, integer edge)    {        // only change linear motor if one of the
linear controls are pressed                                                          vector motor;
   integer motor_changed = level & gLinearControls;
        if (motor_changed)
        {
            if(level & CONTROL_UP) //CUBEY
            {
                if (gLinearMotor.x < 0)
                {
                    gLinearMotor.x = 0;
                }
                else if (gLinearMotor.x < 30)
                {
                    gLinearMotor.x += 5;
                }
                motor_changed = TRUE;
            }
            if(level & CONTROL_DOWN) //CUBEY
            {
                if (gLinearMotor.x > 0)
                {
                    gLinearMotor.x = 0;
                }
                else if (gLinearMotor.x > -30)
                {
                    gLinearMotor.x -= 5;
                };
            motor_changed = TRUE;
         }
          llSetVehicleVectorParam(VEHICLE_LINEAR_MOTOR_DIRECTION, gLinearMotor);
      }
      // only change angular motor if the angular levels have changed        motor_changed =
(edge & gOldAngularLevel) + (level & gAngularControls);
      if (motor_changed)
      {
         motor = <0,0,0>;
         if(level & (CONTROL_RIGHT|CONTROL_ROT_RIGHT))
         {
            // add roll component ==> triggers banking behavior
            motor.x += 10;
         }
         if(level & (CONTROL_LEFT|CONTROL_ROT_LEFT))
         {
            motor.x -= 10;
         }
         if(level & (CONTROL_BACK)) // CUBEY
         {
            // add pitch component ==> causes vehicle lift nose (in local frame)
            motor.y -= 8;
         }
         if(level & (CONTROL_FWD)) // CUBEY
         {
            motor.y += 8;
         }
          llSetVehicleVectorParam(VEHICLE_ANGULAR_MOTOR_DIRECTION, motor);
      }
      // store the angular level history for the next control callback
      gOldAngularLevel = level & gAngularControls;
   }
}
```