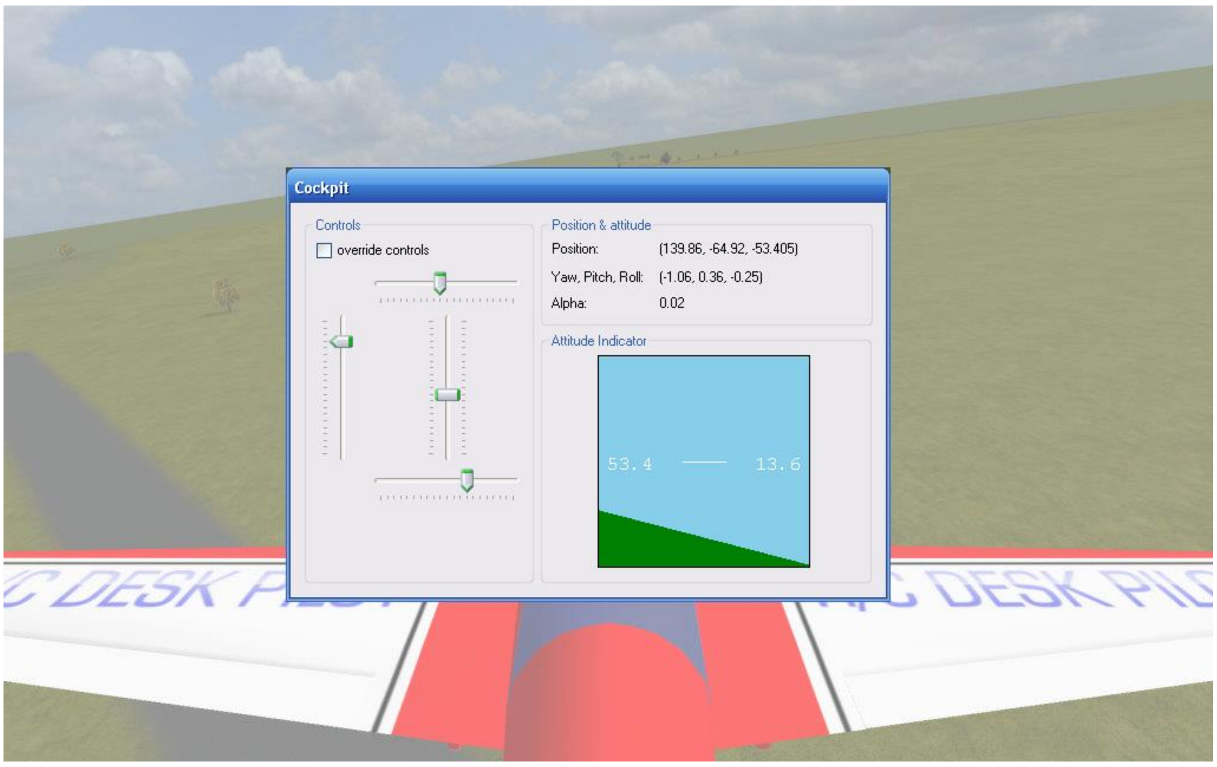
Red Cross RC Sim

0.1.1 SDK Manual



Position: (139.86, -64.92, -53.405)
Yaw, Pitch, Roll: (-1.06, 0.36, -0.25)
Alpha: 0.02

Introduction

Red Cross RC Sim is a free, open-source RC flight simulator.

Red Cross RC Sim comes with an SDK that allows developers to write their own plug-ins for the sim. Version 0.1.1 allows a developer to access all aircraft and control parameters, as well as override the most crucial parts of the flight model. This should make it possible to

- implement and experiment with your own flight model (and basically use the simulator as a visualization tool);
- integrate the simulator with other software, e.g. autopilot/UAV control software.

Contents of the SDK

If you look at the installation folder of Red Cross RC Sim (by default c:\program files\RC Desk Pilot), you'll see a folder named "SDK". Open it up, and you'll find:

- RCDeskPilot.API.dll: the .NET assembly containing the API interface
- RCDeskPilot.API.sln: a Visual Studio 2008 solution file containing sample code
- RCDeskPilot_API_manual.pdf: this manual

Getting started

Running Red Cross RC Sim with a plugin

The version of Red Cross RC Sim that you've installed together with this SDK already contains a compiled version of the sample code that's included. If you run Red Cross RC Sim, it should open a second "cockpit" window. This cockpit window is in fact part of the plugin for which the code is included with the SDK.

Getting the simulator to use a plugin is pretty simple. Open up the file "frameworkconfig.xml" in a text-editor (or Visual Studio). This is an XML file containing the sim settings. You'll find an XML node:

```
<Application.KeyValues>
  <Key>ApiFlightModel</Key>
  <Value>RCDeskPilot.API.Sample.dll,
         RCDeskPilot.API.Sample.MyFlightModel</Value>
</Application.KeyValues>
```

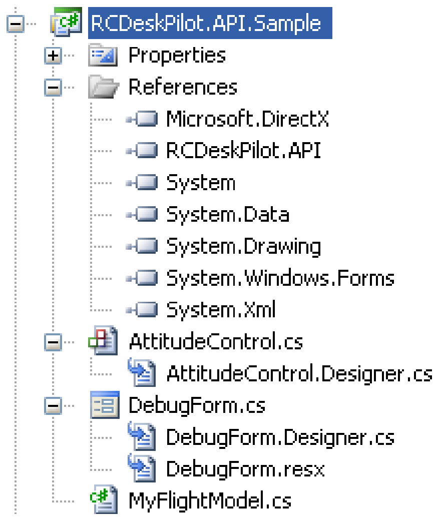
If this node is not present, ot the <Value> node is empty, then no plugin is used. The <Value> node indicates which assembly and which class to use as a plugin flight model. The first part tells the sim to load the RCDeskPilot.API.Sample.dll assembly, the second part (after the ',') which class to load. The assembly is looked for in the main folder of the sim.

Building the sample plugin

The SDK contains the code for a sample plugin demonstrating the most important features. For building it, you'll need Visual Studio 2008 for C#. You can download a free

version called Visual Studio 2008 Express from the website
http://www.microsoft.com/exPress/

Red Cross RC Sim will work with plugins written in any .NET compatible language, but the sample is written in C# 2.0.
When you're ready, open up the solution file RCDeskPilot.API.sln from the SDK folder. This solution contains the RCDeskPilot.API.Sample project with following files and references:



```
RCDeskPilot.API.Sample
    Properties
    References
        Microsoft.DirectX
        RCDeskPilot.API
        System
        System.Data
        System.Drawing
        System.Windows.Forms
        System.Xml
    AttitudeControl.cs
        AttitudeControl.Designer.cs
    DebugForm.cs
        DebugForm.Designer.cs
        DebugForm.resx
    MyFlightModel.cs
```

You'll notice that the project references RCDeskPilot.API.dll. This is the assembly containing the actual API definitions. The project also references Microsoft.DirectX. Red Cross RC Sim is built using the managed version of DirectX 9.0c. It is used in the API because it contains some useful types and functions, like the 3-dimensional Vector3 structure.

The project contains 3 files each with a different purpose:

- MyFlightModel.cs: this is the class that really matters. This actually defines the plugin. More about that later
- DebugForm: this is the "cockpit" form that you've seen when running the sim. It contains the Windows.Form code for displaying the debug information.
- AttitudeControl: this is a simple custom control that remotely resembles an attitude indicator.

Now for the actual building: simply press the ▶ Debug button to compile the plugin.

If it worked, you should see a bin\debug folder appear inside your project folder. This will contain the compiled plugin as "RCDeskPilot.API.Sample.dll". You can copy this file over to the sims main folder, overwrite the existing one and test with your own compiled plugin. You are now ready to start coding!

Writing your own plugin

We'll start with the sample plugin. Open up MyFlightModel.cs.

We'll go over the specific parts of this file.

```
using RCDeskPilot.API;
```

This tells the compiler to look in the API namespace for symbols.

```
public class MyFlightModel : FlightModelSimple
```

This is the most important line: it defines a class "MyFlightModel" that derives from a baseclass FlightModelSimple from the RCDeskPilot.API namespace. This provides our class access to a whole lot of useful data from the sim. It also tells the sim that this class can be used as a plugin. This is the classname that you'll find in the frameworkconfig.xml file.

By overriding methods from its baseclass, you can define which parts of the flightmodel you want to implement yourself.

```
public override void Initialize()
{
    debugForm = new DebugForm();
    debugForm.FlightModel = this;
    debugForm.Show();
}
```

The Initialize() method is called each time a new flightmodel is being initialized. This normally only happens when a new airplane has been selected. This method is called on the main thread of the program. This is important if you want to do stuff with Windows Forms. Some methods will be called in a seperate thread and you must never do Forms stuff from any thread other than the main thread (.NET 2.0 will give you an exception if you try to anyway). So in this example, the cockpit debugform is created.

```
public override void ShutDown()
{
    if (debugForm != null)
    {
        debugForm.Close();
        debugForm.Dispose();
        debugForm = null;
    }
}
```

The ShutDown() method is called when the flightmodel is being shutdown. You can use this call to clean up any resources that you've created (in this case, the cockpit form).

```
  public override void UpdateControls()
  {
    if (debugForm != null)
    {
      if (debugForm.OverrideControls)
      {
        this.Ailerons = debugForm.Ailerons;
        this.Elevator = debugForm.Elevator;
        this.Rudder = debugForm.Rudder;
        this.Throttle = debugForm.Throttle;
      }
    }
  }
```

This is the call that will be useful for doing autopilot testing: it gives you the possibility to override the controls that the sim itself has detected. You can test it by checking the "override controls" checkbox of the sample plugin. As you see in the code above, the 4 control input values from the cockpit form will be used instead of the default ones. This method is called on the main thread, once every frame.

```
public override bool CalculateForces(float elapsedTime,
            Microsoft.DirectX.Vector3 wind)
{
  // Here you've got access to all parameters from the flightmodel,
  // as well as the aircraft parameters (through the
  // AircraftParameters property).

  // Fx = force front/back
  // Fy = force left/right
  // Fz = force up/down

  // Return true if you want to override the default implementation
  // Return false if you don't want to implement this method yourself.
  return false;
}

public override bool CalculateTorques(float elapsedTime,
            Microsoft.DirectX.Vector3 wind)
{
  // Here you've got access to all parameters from the flightmodel,
  // as well as the aircraft parameters (through the
  // AircraftParameters property).

  // Tx = Torque around x-axis.
  // Ty = Torque around y-axis.
  // Tz = Torque around z-axis.

  // Return true if you want to override the default implementation
  // Return false if you don't want to implement this method yourself.
  return false;
}
```

The motion of the aircraft is determined by to main contributions to the flightmodel: forces and torques. The forces determine the accelerations (and hence speed and location) that the aircraft will experience. The torques determine the angular accelerations that will in turn determine the attitude of the aircraft. You can override each one of these as you see fit. The result of these methods should be assigned forces Fx, Fy, Fz or Tx, Ty, Tz respectively.

For the calculation of these forces and torques you can use any of the available properties or values available to you by deriving the SimpleFlightModel class. However, changing any of these properties (besides forces and torques) may have undesireable consequences and should not be attempted, nor needed.

Remark that the forces and torques are to be calculated in a coordinate system relative to the aircraft. All coordinate transforms are performed by the underlying code.