Kerbal Krash System mods

Writing your own mods that use KKS

Enzo Meertens
Document version 0.1

Introduction

Kerbal Krash System

What in the name of Wernher von Kerman is Kerbal Krash System?

Kerbal Krash System (KKS) is a mod for Kerbal Space Program (https://kerbalspaceprogram.com). This mod replaces explosions with malfunctions and deformations (and a few explosions) based on the velocity (RelativeVelocity) and the contact point (ContactPoint) of the impact.

ToleranceScaling

To be able to notice deformations and malfunctions, the original crash tolerance speeds (***OCTS***) of parts are scaled by a certain value based on the type of the part. E.g.: Container parts will have their crash tolerance speeds increased by a factor of **5.0**[1]. If parts exceed their scaled crash tolerance speeds (***SCTS***), they will still explode.

Parts only receive damage and deformations when impacting other objects at speeds exceeding their *OCTS*.

Malleability

Some parts have very high *OCTS*[2] which causes deformations to only function above this threshold. Therefore a ***malleability*** variable is added. *Malleability* indicates the ease of deforming and damaging. E.g. The **Mk1-2 Command Pod's** original crash tolerance speed is 45m/s. Using a *ToleranceScaling* of 2.0 will scale this speed to 90m/s. This *SCTS* (90m/s) is then scaled using the *malleability* variable (of value 4.0) during the actual damage calculation to bring this *SCTS* value down to 22.5. So if this Mk1-2 Command Pod impacts Kerbin at a velocity (**v** in m/s) of **22.5 < v < 90**, the Command Pod will take damage without exploding.

KKS-mods

What's a KKS-mod?

KKS-mods are mods based on KKS. KKS-mods use the "**OnDamageReceived**" and "**OnDamageRepaired**" events to influence Kerbal Space Program gameplay. These influences can be anything: explosions based on damage, leaks, disabling/enabling certain events (the ability to do conduct science or the ability to go EVA), influence efficiency of parts based on damage, etc.

KKS-mods will most likely be the solution to almost all community requested features for KKS.

---

[1] Version 0.25

[2] Some command modules exceed original crash tolerance speeds of >35m/s

Usage

Setting up your project

Requirements

- Visual Studio (2013/2015)
- Kerbal Space Program
- Kerbal Krash System mod
- *Module Manager* (not needed, but extremely useful)

Targeting the correct framework version (.NET 3.5)

In Visual Studio go to the top-left toolbar and click "*Projects*" > *"<Project name> Properties…"*. This will open up the properties page. Navigate to *"Application"* and set *"Target framework"* to **.NET Framework 3.5**. This compiles your code using the same framework version as Unity.

Adding required references

In Visual Studio go to the top-left toolbar and click "*Projects*" > "*Add Reference…*". This will open up the Reference Manager dialog. To add a reference to Kerbal Krash System, go to "Browse" and browse to your KKS installation folder:
(**<KSP>/GameData/KerbalKrashSystem/Plugins/KerbalKrashSystem.dll**). This will add the reference to your project.

Next to KerbalKrashSystem, you also need the **UnityEngine.dll** and the **Assembly-CSharp.dll** found in **<KSP>/KSP_Data/Managed/**.

Now you're ready to write your first KKS-mod!

Writing your first KKS-mod

The following code snippets will attempt to demonstrate a few basic KKS-mods.

Repairing damage KKS-mod

Code

This KKS-mod adds a button to the menu displayed when right-clicking a part in EVA and in range of the part. Important bits are bolded and underlined.

```
using KKS; //Indicate you want to use functions/variables located in KerbalKrashSystem.dll.

... //Other usings and namespace.

//Try to maintain the same naming convention for KKS-mods.
public class ModuleKerbalKrashSystem_Repair : PartModule
{
    //Variable to hold our KerbalKrashSystem component.
    private KerbalKrashSystem kerbalKrash;

    /// <summary>
    /// Called every time this part gets enabled, so this should always get executed.
    /// </summary>
    private void OnEnable()
    {
        //Get the KerbalKrashSystem component (script) attached to this part.
        kerbalKrash = part.GetComponent<KerbalKrashSystem>();
    }

    /// <summary>
    /// Right-click repair button event: repairs the last applied damage.
    /// </summary>
    //[KSPEvent] indicates the following function is executable from the right-click menu.
    [KSPEvent(guiName = "Repair", active = true,  guiActive = true, externalToEVAOnly = true,
        guiActiveEditor = false, guiActiveUnfocused = true, unfocusedRange = 3.0f)]
    public void Repair()
    {
        //No krashes to repair.
        if (kerbalKrash.Krashes.Count == 0)
            return; //Skip the rest of this function.

        //Fully repair the part.
        kerbalKrash.Repair();

        //Remove the last krash.
        kerbalKrash.Krashes.RemoveAtkerbalKrash.Krashes.Count - 1);

         //Apply all remaining krashes.
        foreach (KerbalKrashSystem.Krash krash in kerbalKrash.Krashes)
            kerbalKrash.ApplyKrash(krash);

        //Inform the user about the repaired state of the part.
        ScreenMessages.PostScreenMessage("Repaired to " + kerbalKrash.Damage.ToString("P"),
            4, ScreenMessageStyle.UPPER_CENTER);
    }
}
```

You can now compile your code and move the compiled class library file (in this case "**KerbalKrashSystem_Repair.dll**" to your **<KSP>/GameData/KerbalKrashSystem/Plugins/KKS-mods/KerbalKrashSystem_Repair/Plugins/** folder.

You will have to add some of these folders yourself.

Adding to parts (using Module Manager)

We will use Module Manager to add this module to every single part containing the KerbalKrashSystem module. Module Manager uses configuration (.cfg) files to load/unload modules.

To add this module to every single part containing a KerbalKrashSystem module, we will add a configuration file called "**KerbalKrashSystem_Repair.cfg**" to our **<KSP>/GameData/KerbalKrashSystem/Plugins/KKS-mods/KerbalKrashSystem_Repair/Configs/** folder.

To allow Module Manager to do its job, we will add the following code to the configuration file:

```
@PART[*]:AFTER[KerbalKrashSystem]:HAS[MODULE[ModuleKerbalKrash*]]:FOR[KerbalKrashSystem_Repair]
{
	MODULE
	{
		name = ModuleKerbalKrashSystem_Repair
	}
}
```

Important note: Do not use spaces or line-endings in the Module Manager operation line (first line).

This will add a **module** of type **ModuleKerbalKrashSystem_Repair** to every part (**@PART[*]**) that has a module called **ModuleKerbalKrash*** [3] attached to it, after *KerbalKrashSystem* has been loaded. KerbalKrashSystem is the main module and has to be loaded first in most cases.

FOR[KerbalKrashSystem_Repair] is an indicator, this will allow functions such as **AFTER**[KerbalKrashSystem_Repair] to function properly.

[3] The ***-operator is used to indicate zero to many wildcard characters. For "ModuleKerbalKrash***" this means every module name including "ModuleKerbalKrash" will get used (ModuleKerbalKrashSystem_Engine, ModuleKerbalKrashSystem_Foo_Bar, etc.).

Influencing Gameplay events

Code

This KKS-mod adds special events to all engines (containing the *ModuleEngines* module). Important bits are bolded and underlined.

```
//To access crash tolerance scaling and malleability of parts inherit from KerbalKrashSystem.
public class ModuleKerbalKrashSystem_Engine : KerbalKrashSystem
{
    public float OverheatScaling = 10.0f;

    private float originalHeatProduction;
    private ModuleEngines engine;

    protected override void OnEnabled()
    {
        base.ToleranceScaling = 8.0f;
        base.Malleability = 2.0f;

        //Get the module containing all engine data.
        engine = (ModuleEngines)part.GetComponent(typeof(ModuleEngines));

        originalHeatProduction = engine.heatProduction;

        //Register the DamageReceived and DamageRepaired event handlers.
        //Both events use the same parameters, so we can use the same function.
        DamageReceived += ModuleKerbalKrashEngine_DamageReceived;
        DamageRepaired += ModuleKerbalKrashEngine_DamageReceived;
    }

    protected override void OnDisabled()
    {
        //Unregister the DamageReceived event handler.
        DamageReceived -= ModuleKerbalKrashEngine_DamageReceived;
        DamageRepaired -= ModuleKerbalKrashEngine_DamageReceived;
    }

        /// <summary>
        /// This event handler function is called when this part gets damaged.
        /// </summary>
        /// <param name="damage">New damage percentage.</param>
    private void ModuleKerbalKrashEngine_DamageReceived(KerbalKrashSystem sender, float damage)
    {
            //Increase this engine's heat production when damaged.
            //If damage equals zero, the original heat production will be used.
        engine.heatProduction = originalHeatProduction +
             (float)(part.maxTemp * damage * OverheatScaling);
    }
}
```

You can now compile your code and move the compiled class library file (in this case "**KerbalKrashSystem_Repair.dll**" to your **<KSP>/GameData/KerbalKrashSystem/Plugins/KKS-mods/KerbalKrashSystem_Engine/Plugins/** folder.

You will have to add some of these folders yourself.

We will use Module Manager to add this module to every part containing the *ModuleEngines* module.

To add our ModuleKerbalKrashSystem_Engine to every part containing the *ModuleEngines* module, we will create a configuration (.cfg) file for Module Manager.

We name our configuration files something recognizable and consistent, e.g.: "**KerbalKrashSystem_Engine.cfg**" and add it to the **<KSP>/GameData/KerbalKrashSystem/Plugins/KKS-mods/KerbalKrashSystem_Repair/Configs/** folder.

To allow Module Manager to do its job, we will add the following code to the configuration file:

```
@PART[*]:HAS[@MODULE[ModuleEngines*]]:FOR[KerbalKrashSystem_Engine]
{
	MODULE
	{
		name = ModuleKerbalKrashSystem_Engine
	}
}
```

Important note: Do not use spaces or line-endings in the Module Manager operation line (first line).

This will add a **module** of type **ModuleKerbalKrashSystem_Engine** to every part (**@PART[*]**) that has a module called **ModuleEngines***[4] attached to it.

FOR[KerbalKrashSystem_Engine] is an indicator, this will allow functions such as **AFTER**[KerbalKrashSystem_Engine] to function properly.

[4] The ***-operator is used to indicate zero to many wildcard characters. For "ModuleEngines***" this means every module name including "ModuleEngines" will get used (ModuleEngines, ModuleEnginesFX, etc.).

You’re done! Run Kerbal Space Program and check out your new and amazing mod!