XML   C# Pseudo Classes   C# Instance

C# Defs are pseudo classes in that they store pseudo-constant data and methods (in themselves, their Properties and their ComponentProperties) as well as create instances of themselves, as determined by the thingClass.

XML Defs

<ThingDef Class="<NameSpace.MyThingDef>"> `xml`
Created to store new types of data in a definition of a thing. Note this can be used for any Def.

ThingDef `xml`
The definition of something placed in the map.

Def `xml`
The definition of something in RimWorld from the XML file. This is a "blueprint" for anything generated in the game.

multiple elements

C# Defs

<MyThingDef>.cs `c#`
If you want to add new tags to an item in XML, you need to extend this (but still call it ThingDef in your XML file).

→ *populates* →

ThingDef.cs `c#`
The data and generic functionality for managing items on the map.

Type thingClass

▷ BuildableDef

Def.cs `c#`
...

▽ Editable   has these properties

List<CompProperties> comps

C# Thing Instance

<MyThing>.cs `c#`
Instantiated by ThingDef when created. ThingDef's thingClass.

either

def →

Thing.cs `c#`
Instantiated by ThingDef when created. ThingDef's thingClass. An instance of a ThingDef in the game. It loads from a ThingDef and a save file in Thing.ExposeData.

Entity

ThingWithComps.cs `c#`
Has Components, a type of Property, which plays an active role in the Thing.

XML Properties

<<element>> `xml`
A generic element that sets data in a property or variable.

<plural element> `xml`
An element that stores many elements, like <li>. This is a way to reference ThingDefs or strings. Ex. <stuffCategories> on items.

C# Properties

<MyTag>Properties.cs `c#`
...

→ *populates* →

<element>Properties `c#`
Store data assigned in ThingDefs and has methods. The elements inside the <element> are to values or other properties.

→ *populates* →

Type `c#`
A variable on the Thing like a float or int or string. ex. <Mass>0.4</Mass>

→ *populates* →

List<ThingDef> `c#`
The c# file creates the list and converts the items in the tag into a ThingDef reference, Properties or just values. ex. Metallic is turned into SuffCategoryDef::Metallic and Tribal into just a string. This occurs based upon the class T in the List<T>.

Components are a mix between a Property and a ThingDef. Components have data in properties and have an instantiated part with methods and local data in the instance.

XML Component Properties

<li Class=' <Namespace.MyComponent>'> `xml`
...

<comps> `xml`
A plural element in the ThingDef XML when thingClass is of type "ThingWithComps". This element populates 'List<ComponentProperties> comps' and instantiates 'List<ThingComp> comp' on a ThingWithComps.

many

<li Class='Component'> `xml`
...

many

C# Component Properties

→ *populates* →

CompProperties.cs `c#`
These store the XML data for a component of a ThingDef. When a Thing is instantiated, they are converted into their ThingComp version inside ThingWithComp.comps. The compClass is used when turning a CompProperties_X into a CompX.

Type compClass

→ *populates* →

CompProperties_X `c#`
Generally, these have a constructor that sets the compClass variable. Additionally, any further static information of the CompProperties is stored here.

either

CompProperties_<My>X `c#`
Your created Components.

C# Component Instance

parent

ThingComp.cs `c#`
A class that stores the instance of a CompProperties for a ThingWithComps. It has its own data but uses the CompProperties's data for defined values that don't change. Also, this is where your create methods for a Component.

← *props* ←

List<ThingComp> comps

CompX.cs `c#`
Child of ThingComp, specializing its behavior for X. Examples of X are CompPower, CompFlickable, etc.

either

Comp<My>X `c#`
Your created Components.