Guide contents

Choose a route

Build the package first

Make one XML content change

Patch an existing Def safely

Add C# only when needed

Test and release

A complete first project: a new plant

Sources / References

Documentation: In progress **Quick start: Complete**

Offline copy from ModDocs. Source links remain clickable.

RimWorld modding

RimWorld supports many content changes through XML definitions (**Defs**). C# assemblies are for behavior that XML cannot express, and Harmony patches are for carefully changing existing code. The RimWorld Wiki modding tutorials cover all three routes; some older pages are marked outdated, so check their version before using an example.

> **Quick start:** make a mod folder with `About/About.xml`, add one small XML Def in `Defs/`, enable the mod, and verify it in a new test game. Add code only once you know the XML route cannot express your idea.

Choose a route

What you want	Begin with	Next reference
New item, recipe, plant, building, research	XML Def	Defs introduction

What you want	Begin with	Next reference
Change existing content without replacing its whole Def	XML patch operation	Tutorial index
New behavior or calculation	C# assembly	Writing custom code
Alter how an existing method runs	Harmony patch	Harmony tutorial

An XML Def supplies data to a game system. It does not automatically create a new kind of behavior. A custom C# class can be linked from XML when the Def exposes a class field or component list; the linking guide explains that connection.

Build the package first

The smallest practical folder has a root mod directory, an `About` folder containing `About.xml`, and any content folders you need:

```text
MyMod/
  About/
    About.xml
  Defs/
    ThingDefs/
      MyContent.xml
  Textures/
  Assemblies/
```

Only create the optional folders you use. The names of special folders are case-sensitive on some platforms; see mod folder structure. The root folder's name is for your organization; `About.xml` carries the mod's identity.

`About.xml` needs a unique `packageId`, a display name, an author, a description, and the game versions you have actually tested. Copy the current field shape from the About.xml guide, then replace the example identity. A duplicate package ID can make one of the mods unusable. Do not claim support for a version you have not tried.

Make one XML content change

Inspect a comparable Def in the installed game's `Mods/Core/Defs` directory. Record its type and required fields, then create your own Def with a distinct `defName`. The XML Defs guide explains the relationship between classes and their data. A basic item may also need a texture path, category, costs, recipes, or research access. Those links vary by type: a loaded Def can still be unobtainable in normal play.

After enabling the mod, check the game's log for parse errors and duplicate definitions. Find the content in game and test the intended route to obtain it. Keep the first change tiny: one working Def teaches more than a large folder of copied XML with many unknown dependencies.

Patch an existing Def safely

If the aim is to adjust base content, prefer a targeted XML patch over replacing a whole vanilla Def. A whole replacement can erase changes from the game or other mods. Make the patch conditional where it depends on a DLC or another mod, and verify behavior with that dependency both enabled and disabled. The wiki's mod folder structure also documents versioned load folders and `LoadFolders.xml` for separating content across game versions.

Add C# only when needed

For behavior, create a C# project using the version of RimWorld and assemblies your target build requires, compile your DLL into `Assemblies/`, then link its class from XML where the Def format supports that field. This is a different task from writing the XML alone; follow solution setup and linking XML and C#. Keep game logic in your assembly and data in XML so each can be checked separately.

Harmony can run prefixes and postfixes around existing methods. It is powerful, but a patch can conflict with other mods or break when the target method changes. Use a precise target, test on the game's current version, and make the patch as small as possible. The Harmony tutorial recommends using the game's shared Harmony dependency rather than bundling a stale copy of `0Harmony.dll` in every mod.

Test and release

1. Load only your mod and its declared dependencies. Fix the first XML or assembly error in the log.
2. Check a fresh colony and a saved colony. A new Def may load while an existing save retains old state.
3. Test with the DLC and other mods your content refers to; check whether load order matters.
4. Verify every visible name, texture, recipe, and research unlock in normal play.
5. Package the mod root with its `About` metadata, source attribution, and only the built files it needs. Retest the packaged folder.

A complete first project: a new plant

The wiki's Basic Plant tutorial follows a complete, connected example instead of an isolated XML tag. It creates a plant, harvest item, and crafting recipe. Its working sample repository is useful for comparing folder structure when a field is unclear. Start by making `About/About.xml` with your own package ID, then add the plant `ThingDef` entries in `Defs/ThingDefs_Plants/`. The tutorial uses separate base, cultivated, and wild definitions; their `ParentName` relationships are important because the inherited fields determine how the concrete plants behave.

Add the harvest item's `ThingDef`, the recipe `RecipeDef`, and every texture referenced by those defs. Keep each Def's `defName` unique. Load the mod alone, read the first XML error, and then test in a new colony: locate the plant in an appropriate biome or sow it, let it grow, harvest the item, and inspect the recipe at its intended workbench. If the plant exists but yields nothing, inspect its product reference; if the item exists but cannot be crafted into the result, inspect the recipe and workbench access. The Def structure tutorial explains why a parsed XML file can still point to a missing Def.

Finally, move the loose development folder out of the active mod path and install a packaged copy. Check the name, both plant textures, harvest, recipe, and log again. The sample tutorial is community-maintained and may lag a new RimWorld version, so compare its XML to the corresponding current base-game Defs before release.

The plant, harvest item, recipe, textures and clean-package test form a complete first XML content mod. The walkthrough is source-reviewed against the linked community

tutorial, not locally run in RimWorld here. Buildings, C# behavior, save-data migrations and version compatibility need their own checks; the tutorial index is the source for those deeper paths and flags several legacy tutorials that need caution.

Sources / References

External project and documentation links cited in this guide. Check the upstream release and version before following version-sensitive steps.

- RimWorld Wiki modding tutorials
- Defs introduction
- Writing custom code
- Harmony tutorial
- linking guide
- mod folder structure
- About.xml guide
- Basic Plant tutorial