

Customizing reMarkable tablets

Claude Code and custom application development

Andreas

Can you determine if Claude Code could help me write a custom application for the reMarkable tablet?

Claude

Yes—and it’s a good fit for how you already work. The reMarkable is a regular ARM Linux device running reMarkable OS (codename Codex), built with Yocto. The main user app, Xochitl, is proprietary and built with Qt/C++, but nothing stops you running your own binary alongside or instead of it. The basic loop is exactly what you’d expect: cross-compile on your x86_64 Linux box, scp to the device, ssh in and run it. reMarkable’s own getting-started example is literally `$CC helloworld.c → scp → ssh ./helloworld`.

Three things make this land well for you specifically:

The official SDK is a Yocto cross-toolchain, and it’s only published with an x86_64 host architecture—which is what your Linux machines are. Claude Code runs on that host, drives the cross-compiler, and can run the scp/ssh deploy step itself as bash commands if the tablet is reachable on the USB network (10.11.99.1) or over wifi. So you can close most of the build-deploy loop inside a Claude Code session.

C++ is the native path. Since the device stack is Qt, you can write Qt Quick (QML) or Qt Widgets apps in C++, which leans directly on your existing strength rather than asking you to pick up Rust. The community frameworks are mature if you want them: libreMarkable (Rust, native refresh support), rmkit, KORreader and Plato as reference ports, and RCU for device management.

On checkability—your usual concern—this is mostly a rung-1 domain but with one honest seam. The cross-compile is compiler-arbitered, and “does the binary execute on ARM” is a hard external check. But the e-paper layer is the catch: drawing isn’t normal framebuffer blitting, it needs the partial/full *refresh* ioctls, and whether your refresh behavior, pen latency, and layout actually look right on the display can only be verified on the hardware. Claude Code can’t see the panel. So the arbiter for the *visual* behavior is you-with-the-device-in-hand, not the compiler—weaker than your desktop UI work where the result is observable in-loop.

One wrinkle worth settling early: access differs by model. RM1/RM2 just need the SSH password from Settings. The Paper Pro requires enabling developer mode first, which forces a factory reset, so back up before you start. If you tell me which one you have (or are considering), I can get concrete about the toolchain version and the framebuffer/input specifics.