Unlocking the Full Potential of WPE to Build a Successful Embedded Product

Mario Sánchez Prada

mario@igalia.com

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT NORTH AMERICA

igalia

About me

- **CS Engineer**, partner of **Igalia**
- **Involvement** in some **Open Source communities**
  - e.g. Chromium, WebKit, GNOME
- Other **work** done in the past:
  - Linux-based OSs (i.e. Endless OS, LitI OS)
  - Maemo (Hildon Application Manager)
  - Samsung Smart TV platform

Currently **coordinating** Igalia's **WebKit team**

About Igalia

- Specialized **Open Source consultancy**, founded in 2001
- **Fully remote**, HQ in **A Coruña** (Spain). **Flat structure**
- **Top contributors** to all the main **Web Engines**
  - WebKit, Chromium, Gecko and Servo
- **Active contributor to other areas and OSS projects**
  - V8, SpiderMonkey, JSC, LLVM, Node.js, GStreamer, Mesa, Linux Kernel...
- **Members of several working groups**:
  - W3C, WHATWG, WPT, TC39, OpenJS, Test262, Khronos...

Outline

1. Why do Web engines matter in embedded devices?
2. Common pitfalls using WPE for embedded devices
3. Benefits of a tighter relationship with upstream
4. Best practices for successful integration
5. Real-world case studies
6. Wrapping up
7. Q&A

Why do Web engines matter in embedded devices?

What is a Web engine?

Software component that leverages the power of the **Web Platform**

- **Fetches** HTML / CSS / JavaScript content from multiple sources
- **Interprets** the web content to create an internal representation
- **Produces a result** that users can **interact with**
- It's an **extremely flexible platform**. Examples:
  - Textual and non-textual content
  - Multimedia playback
  - Fully fledged applications

Most popular Web engines:



What is WebKit?

- Open Source **Web engine**, released under **permissive licenses**
- **Main features**:
  - 🕸️ **Complete implementation** of the Web Platform
  - 🚀 **Performance** and **stability**
  - 🔒 **Privacy** and **security**
  - 🔌 **Embeddable** as a top priority (i.e. stable public API)
- **Cross-platform support**:
  - **Desktop & Mobile**: Mac, iOS, Windows, Linux, Android (WIP)
  - **Embedded devices**: set-top-boxes, video game consoles, in-flight entertainment, smart home appliances, GPS devices, digital signage...

https://webkit.org

WebKit Ports

- **WebKit port**: adaptation of WebKit to a specific platform
- **Official WebKit Ports** (*upstream* ports):
  - **Mac**: Safari, Apple Mail, iTunes, App Store...
  - **iOS**: every browser on iOS devices (including Chrome)
  - **Windows**: Microsoft Playwright, Playstation SDK
  - **Playstation**: Playstation 4 & Playstation 5
  - **Linux**: WebKitGTK (GTK apps) and WPE (embedded devices)
    - **Common parts**: GLib, libsoup (networking), GStreamer (multimedia)...
    - **Key differences**: graphics stack, input handling. Different use cases

https://docs.webkit.org/Ports/Introduction.html

What is WPE?

- **WPE is** a **WebKit port optimized** for **embedded devices**
  - Big focus on **flexibility**, **performance** and **security**
  - **Backends-based architecture** and **minimal** set of **dependencies**
  - **Low memory** and **storage footprint**
  - **HW-accelerated graphics** and **multimedia**
  - **Actively maintained** upstream (e.g. up-to-date security fixes)

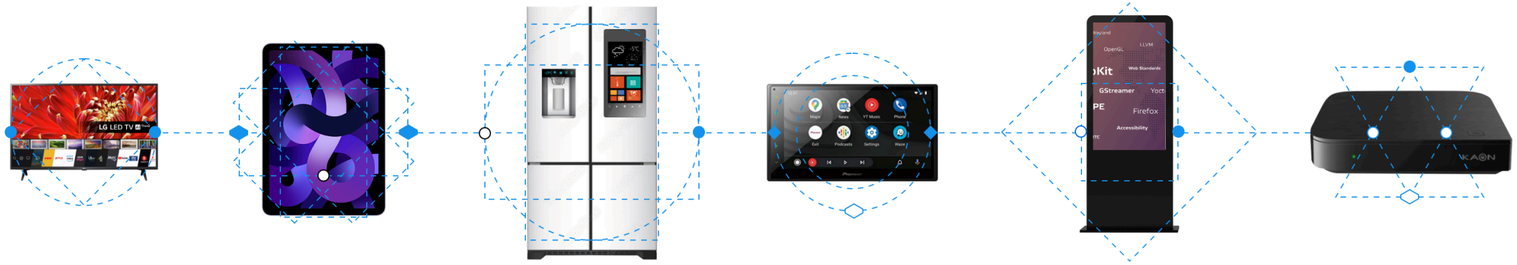
WPE does not rely on any **UI Toolkit** and can also be useful for **less conventional use cases** (e.g. server-side rendering, headless mode...)

https://docs.webkit.org/Ports/Introduction.html

WPE-based products

Some **examples of use cases** we are aware of:

- Set-Top-Boxes
- Smart home appliances
- GPS navigation devices
- Video/Audio conferencing
- Digital Signage
- HiFi sound systems
- Audio streaming
- Headless server-side rendering
- QA and testing
- ...



Why do Web engines matter in embedded devices?

- **Strategic role in the software stack** of embedded devices
  - Rendering, networking, security sandbox, media, I/O, accessibility...
- The **Web platform allows building all sorts of applications**
  - Flexibility for designing, implementing and testing your product
- **Known development stack**
  - Massive pool of web developers that could implement applications
- **Useful to implement all kind of products**
  - Smart home, In-Vehicle/Flight Entertainment, digital signage...

However, **using a Web engine** *effectively* is **more than just fixing bugs**...

Common pitfalls using WPE for embedded devices

Common pitfalls using WPE for embedded devices

- **Treating** WPE as a "**black box**" Web engine
- **Infrequent rebases** and **heavy patching downstream**
- **Delayed feedback cycles** with upstream
- **Not aligning product goals** with upstream

All these situations create **technical debt**, make **integration harder**, affect **development efficiency** and **increase maintenance cost**

WPE as a “Black Box” Web Engine

- **Problem**: Treat WPE blindly as a plug-and-play black box
- **Risks**:
  - Missed opportunity for optimizations
  - Duplicated effort solving issues already handled upstream
- **Possible solutions**:
  - Allocate time for developers to explore the WPE stack
  - Allocate time for developers to contribute back upstream

Why this matters: Properly understanding the Web engine turns WPE into a **strategic advantage** rather than into a hidden liability

Infrequent Rebases and Heavy Patching Downstream

- **Problem**: Downstream patches accumulate, delta becomes too big
- **Risks**:
  - Integration becomes problematic and time-consuming
  - Development work often too focused on bug-fixing
  - Reduced capacity to work on strategic features
- **Possible solutions**:
  - Rebase against upstream as often as possible
  - Contribute well-scoped patches promptly
  - Ensure good downstream practices (e.g. drop patches already upstream)

Why this matters: Frequent syncing avoids complex rebases, **improves the integration process** and **prevents security problems**

Delayed Feedback with Upstream

- **Problem**: Feedback reported to upstream is delayed for too long
- **Risks**:
  - Reduced ability to get proper support from the community
  - Often leads to duplicated work (e.g. issues already fixed upstream)
- **Possible solutions**:
  - Engage in discussions with the community in public channels
  - Report reproducible bugs immediately (i.e. including reduced test cases)

Why this matters: Timely feedback **improves your relationship with upstream** and **reduces the chance of duplicated efforts**

Misaligned Product Goals vs. Upstream Roadmap

- **Problem**: Different goals complicate integration and WPE evolution
- **Risks**:
  - Building bespoke features creates forks that are costly to maintain
  - Forks often require patching in non-upstreamable ways
- **Possible solutions**:
  - Join roadmap discussions upstream to discuss your use-cases
  - Contribute back upstream whenever possible
  - Fund or work on needed features if necessary

Why this matters: Alignment **maximizes efficiency from integrators** and **keeps products on a realistic** and maintainable **upgrade path**

Benefits of a tighter relationship with upstream

Stability & Security

- 🚀 Immediate access to upstream bug fixes
- 🔒 Faster mitigation of security vulnerabilities (CVEs)
- 🔄 Early testing before public disclosures
- 🔧 Lower risk of emergency patching

Performance

- 🔄 Upstream optimizations and performance improvements
- 📈 Changes verified upstream reduce integration risks
- 🎯 Opportunity to prioritize optimizations relevant to your hardware
- 🗺️ Clear visibility into future improvements via upstream roadmaps

Maintainability

- 🔧 Smaller delta with upstream reduces patch maintenance
- 📦 Easier adoption of new WPE releases
- 📅 Predictable long-term maintenance planning
- 💰 Lower ongoing maintenance costs

Alignment

- 📣 Upstream becomes aware of what's relevant for your products
- 🤝 Prioritization upstream aligns better with your business goals
- 🌐 Shared investment in common features with other stakeholders
- 🔮 Build credibility and influence within the WPE community

Community Support

- 🌎 Access to upstream developers and domain experts
- 🧩 Faster identification and resolution of complex issues
- 💡 Shared knowledge base reduces isolated debugging efforts
- 🚀 Build internal expertise via collaboration with upstream

Best practices for successful integration

Open Communication

- **Recommendations**:
  - **Transparency**: Share progress, blockers, and roadmap updates
  - **Share goals**: Collaboratively define goals for your platform integration. Discuss non-standard requirements early to find the best solutions for you
  - **Engage with the community**: e.g. code reviews, general feedback...
- **Benefits**:
  - Prevents divergence with upstream that can complicate maintenance
  - Accelerates problem resolution in collaboration with the WPE maintainers
  - Builds long-term relationships with the upstream community

Frequent Rebasing

- **Recommendations**:
  - **Keep smaller deltas and rebase as often as possible**: avoid complex integrations and potential bugs caused by misalignment with upstream.
    - Faster access to features & fixes (e.g. security fixes)
    - Simpler debugging (e.g. easier bisecting)
  - **Development vs. product branches**:
    - *Tip of Tree (ToT)*: Use as baseline for ongoing feature development
    - *Stable*: Base product releases on stable upstream tags
- **Benefits**:
  - Minimizes maintenance effort (i.e. lower technical debt)
  - Simplifies integration of new releases
  - Enables faster innovation

Contribute back upstream

- **Recommendations**:
  - **Use issue trackers**: Document bugs, enhancements, and discussions
  - **Contribute merge requests**: Enable reviews and feedback from the start
  - **Document decisions**: Provide context for design and architecture choices
- **Benefits**:
  - Higher quality of patches through open reviews
  - Faster identification of possible alternative solutions
  - Shared ownership of the codebase

Upstream-Friendly Commit Practices

- **Recommendations**:
  - **Small, atomic changes**: Easier and faster to review, test, and backport
  - **Upstream-first mindset**: i.e. avoid *hacks*, always consider upstreaming
  - **Clear commit messages**: Explain *what* a patch does and *why* it's needed
- **Benefits**:
  - Simplifies the review process and increases acceptance rates
  - Improves troubleshooting, bugfixing and debugging
  - Builds trust and collaboration with WPE maintainers

Test Automation and CI

- **Recommendations**:
  - **Regression detection**: Automated regression and performance testing
  - **Pre-integration testing**: Validate patches before merging
  - **Upstream tracking**: Automatic testing of upstream snapshots with your downstream patches to detect early possible integration conflicts
- **Benefits**:
  - Prevents breakage caused by upstream changes
  - Enables developing with confidence and fewer regressions
  - Ensures good stability and quality of the end product

Real-world case studies

Real-world case studies

- 🚨 **Case Study #1**: company that maintained a big fork of WPE
- 👍 **Case Study #2**: company that stayed close to upstream
- 🚀 **Case Study #3**: companies working exclusively upstream

🚨 Case Study #1: company that maintained a big fork of WPE

- **Context**:
  - Lots of downstream-specific changes on top of upstream WPE
  - Uses WPE upstream stable releases as base to add their changes on top
  - Rarely contributes patches upstream, often not following best practices
  - Integrates newer versions once every 1-2 years (i.e. skips some of them)
- **Challenges faced**:
  - Painful integration process when moving to newer versions
  - Difficult to innovate and keep up with security patches
  - Difficult to obtain good support from the community
  - Complex alignment of priorities with upstream

Too much effort devoted to **maintaining the fork and fixing bugs**, insufficient allocation of resources to feature development

👍 Case Study #2: company that stayed close to upstream

- **Context**:
  - Downstream changes only for patches not yet upstreamed, or too specific
  - Development on the main branch, stable branches for stabilization only
  - Contributing patches upstream is part of their development process
  - Rebases early and often (e.g. every 2-3 weeks) enabled by automated CI
- **Success story**:
  - Delta with upstream kept to a minimum, integration becomes easier
  - No duplicated efforts, no unnecessary workarounds or hacks
  - Product stabilization aligns with upstream stabilization
  - Upstream-first mentality helps align business goals with upstream

Some **downstream work** still needed but **limited to specific needs**.

Better alignment with upstream and **more time for feature work**

🚀 Case Study #3: companies working exclusively upstream

- **Context**:
  - Some companies don't require downstream work (i.e. can work upstream)
  - Many types of projects possible: implementing a Web spec, performance improvements, new APIs, support for more platforms or more use cases...
  - Great to implement complex features (e.g. CSS Grid, new SVG engine...)
- **Benefits of working directly upstream**:
  - No downstream delta, simpler integration (e.g. product stabilization)
  - Contributing back upstream is a natural part of the process.
  - Upstream won't break your feature (i.e. full integration with upstream CI)
  - Improving technology benefits everyone while supporting specific needs

Ideal way of collaboration from a community standpoint, full transparency and engagement upstream **maximizes efficiency**

Wrapping up

How to Use WPE Effectively

- ☑ **Engage with upstream** as much as possible
  - Align your shareable goals with the next upstream releases
  - **Contribute back upstream**, discuss shared goals **in public forums**

- ☑ **Develop your products** on top of the **upstream development branch** and rely on **stable branches for product stabilization** only
  - Update stable releases in products but continue development in the ToT
  - Discard features that are not stable for your next releases

- ☑ Maintain and evolve **automated CI tailored to your product**
  - Automatically **look out for regressions**, have a policy to handle them
  - Automatically **check upstream versions** using your CI
  - Automate **performance testing**

In a nutshell...

Consider **WPE an Open Source platform for the long term** and **embrace upstream collaboration** as much as possible 👌

PS: Do not treat WPE as "just another vendor package" 🙏

Thanks!

Q&A


igalia