WPE

- Wayland - What and Why?
- Webkit for Wayland
- Advantage of WPE over QtWebKit
- A Traditional Approach for Rendering
- Wayland Approach for Rendering
- Wayland - What and Why?
- Why Westeros?
- Westeros Use Case
- Why WPE
- WPE Architecture
- WPE and Wayland Clients
- Code Restructure
- Work Accomplished in RDK
- Functional Test (HTML5): QTWebkit Vs WPEWebKit
- Functional Test(CSS3): QTWebkit Vs WPEWebKit
- Repositories
- WPE Support with RDK - Recipes
- WPE Support with RDK - Build
- How to work with WPE

Wayland - What and Why?

- Primarily a protocol for compositionof different surfaces/layers.
- compositor + display server + window manager.
- Uses EGLto avoid OS specific windowing functionality.
- Started as a replacement for the X Windowing System (X11).
- Unlike X has no rendering API.
- Its implementations are lightweightwith a small footprint
- Wayland is primarily focused on performance, code maintainability and security

Webkit for Wayland

- Wayland provides simple, elegant, graphics compositing integration between different layers using EGL (interface between Khronos rendering APIs)
- Wayland started as a replacement for the X Windowing System (X11).
- Wayland is not an implementation but a protocol specification between a display server and its clients
- Its implementations are lightweight with a small footprint
- Wayland is primarily focused on performance, code maintainability and security

Advantage of WPE over QtWebKit

Newer WebKit= newer features:

- Contributions in the “upstream” direction are also important to the growth of the RDK, to attract more development and grow the community.
- It’s much faster then QT.
- For Gstreamer, we contributed a way to expose the video fragments, e.g. MPEG DASH smooth streaming, from the browser using Media Source Extensions (MSEs).

Faster:

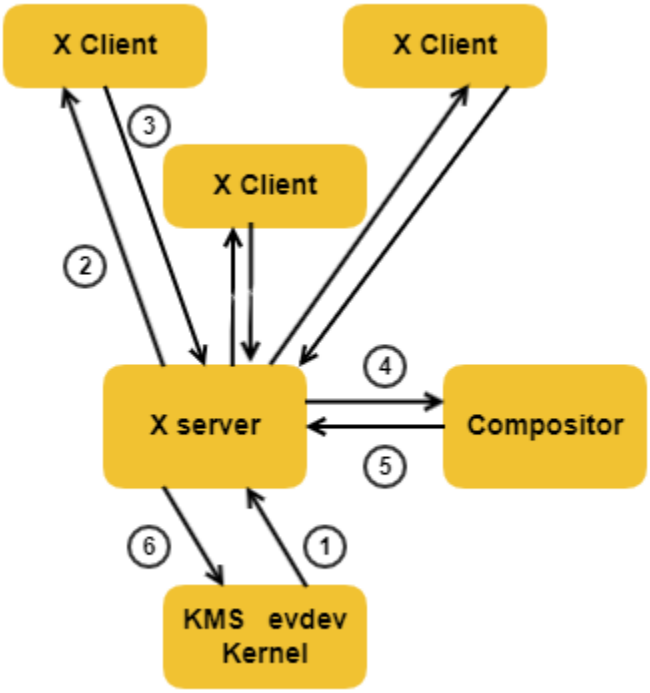
- WebKit2 w/ Threaded compositor vs. QT single thread WebKit1 architecture.
    - UI Process, Web Process, Network Process and Database Process
- New JavaScript Core optimization feature: “Faster Than Light” JIT
- JSC inline with modern day HTML5 development (e.g. better JIT support for closures)

Smaller:

- Compared to QTWebkit, WPE is about 30% smaller — around 22 Megabytes,
- On average, compared to 32 Megabytes while packing 60% more features

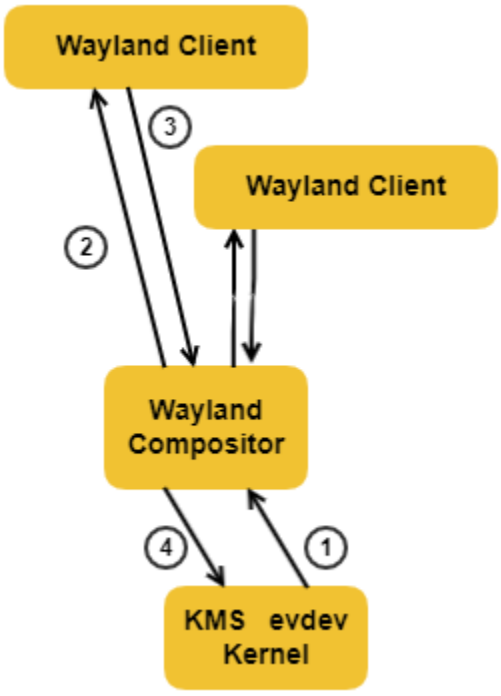
A Traditional Approach for Rendering

In traditional approach, the central role of X-Server and the steps required to get contents on to the screen is presented in below diagram.



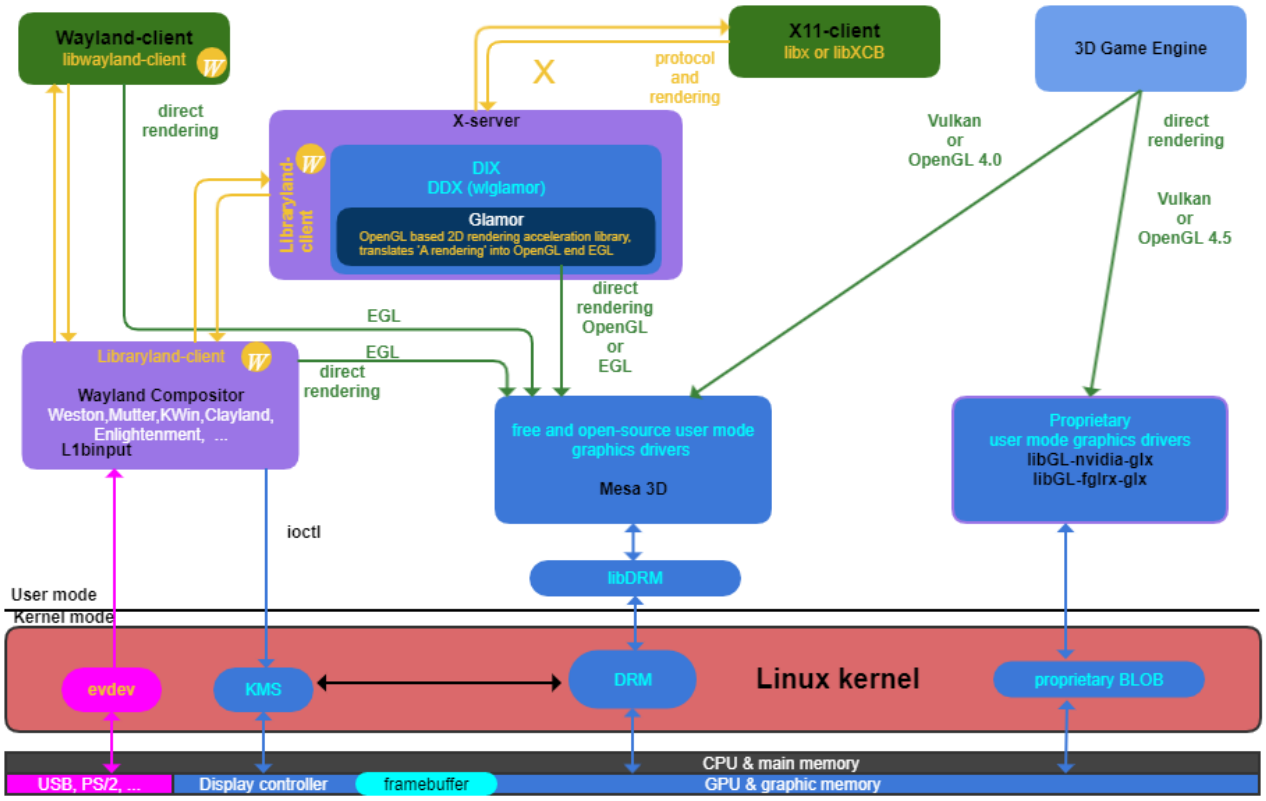
Wayland Approach for Rendering

Removed X-Server and the compositor is the display server. Lets the compositor send the input event directly to the client and lets the client sent the damage event directly to the compositor.

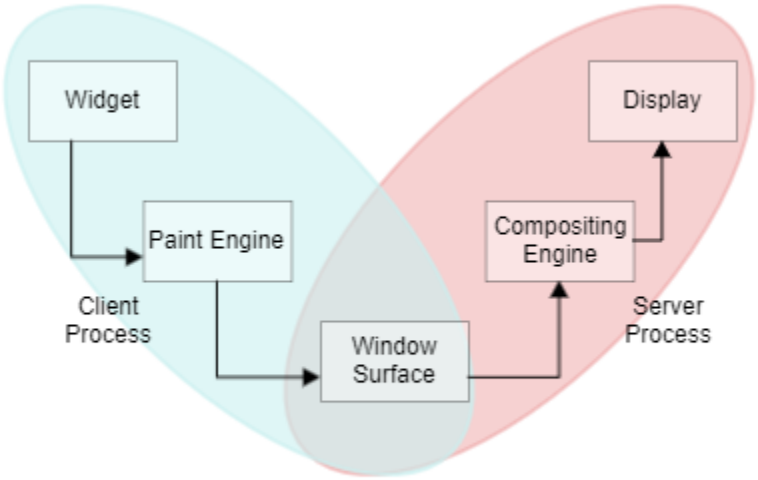


Wayland - What and Why?

“Wayland is a protocol for a compositor to talk to its client as well as a C lib implementation of that protocol”. Weston is the reference implementation for wayland. Westeros is a compositor, a replacement of Weston. This provides clear interface towards graphics and input.



Wayland Process flow:



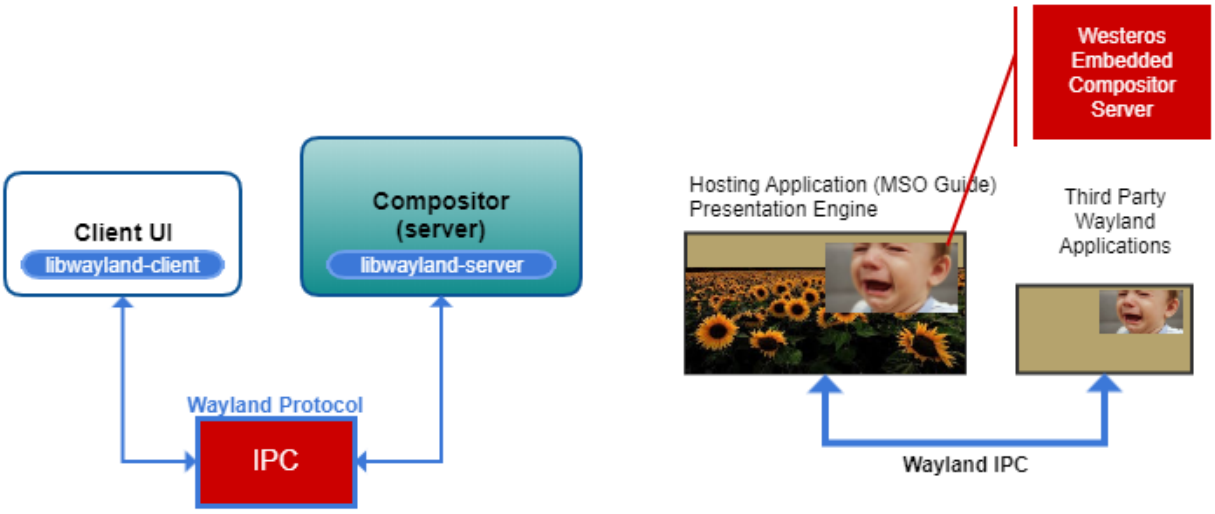
Why Westeros?

- Westeros is smalland simple.
- Being small, easier to understand and maintain.
- Caters to needs of embedded systems over traditional desktop computing.
- Shared library that provides an APIfor creating and operating a compositor.

- Use the included sample compositor app OR implement a custom compositor.
- Allows an application to create a Wayland display within itself to create what we call an embedded compositor.
- Main UI can then control application windowandlifecycle.

Westeros Use Case

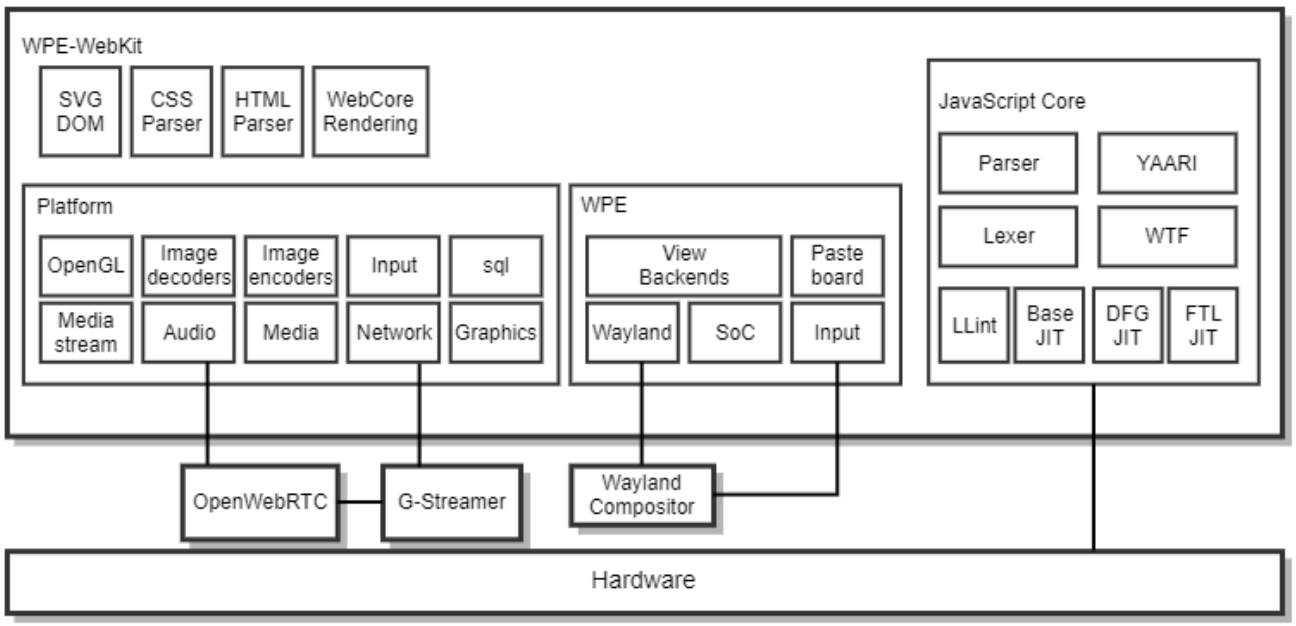
Gives the hosting application (MSO Guide) control over the presentation and composition of 3rd party applications.



Why WPE

- Newer WebKit= newer features
- WebKit2 w/ Threaded compositor vs. QT single thread WebKit1 architecture.
- New JavaScript Core optimization feature: “Faster Than Light” JIT
- Compared to QTWebkit, WPE is about 30% smaller —around 22 Megabytes.

WPE Architecture



WPE and Wayland Clients

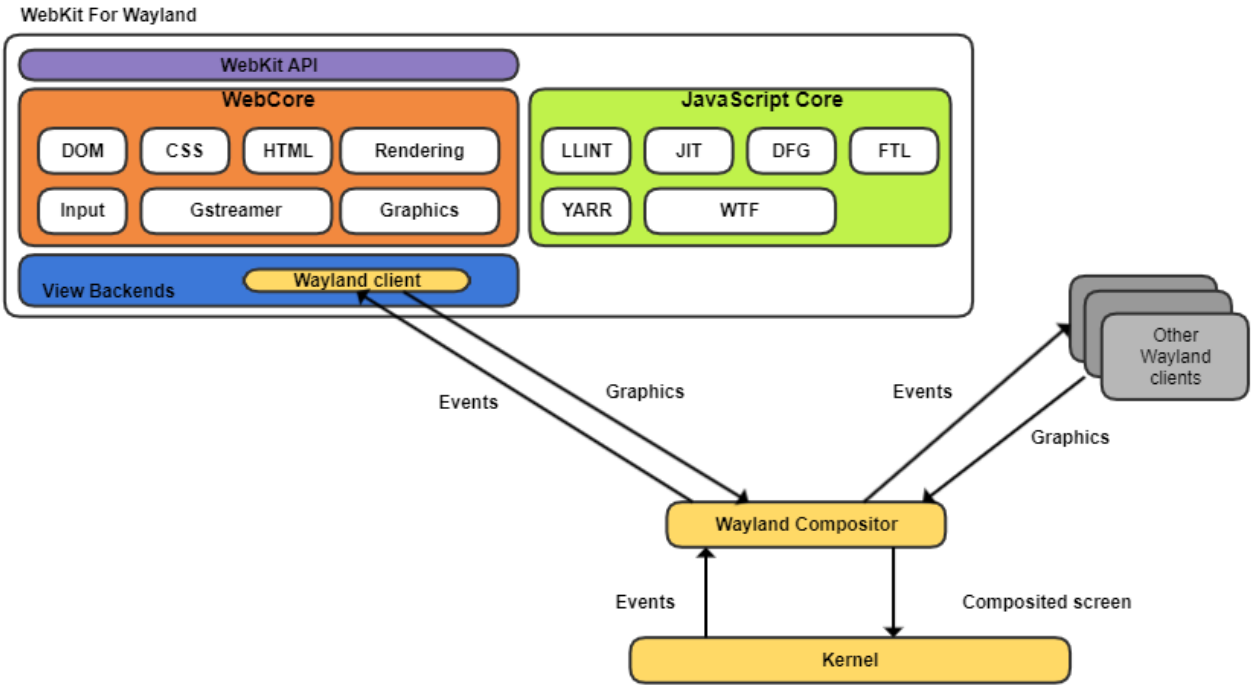
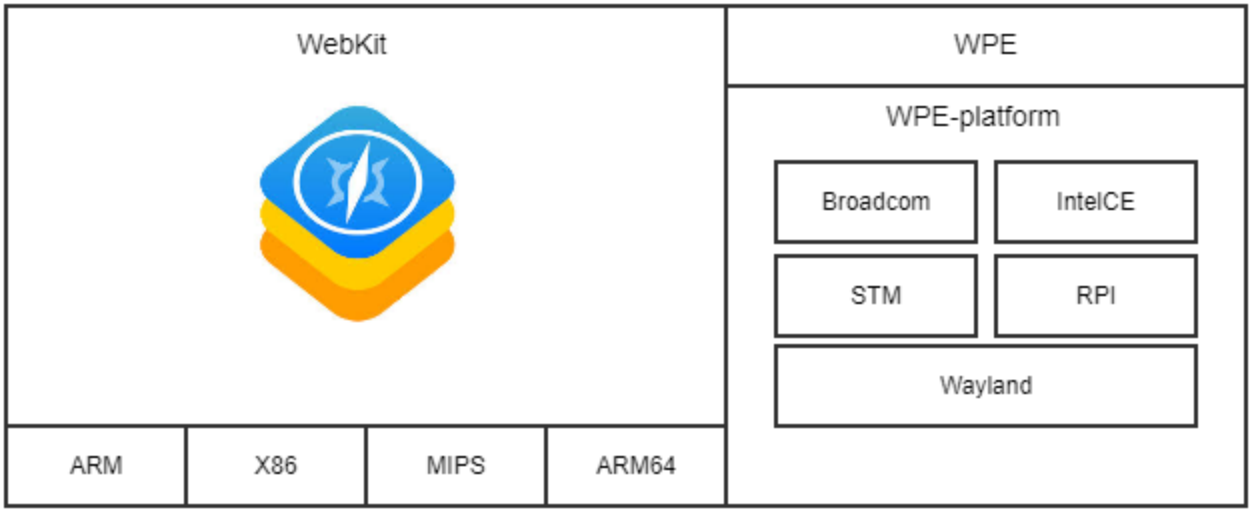


Figure:Webkit for Wayland architecture

Code Restructure

- Code move:
- Source / WPE —> ThirdParty/ WPE & ThirdParty/ WPE-platform
- Easier up-streaming
- Port acceptance
- Refactor WPE to C back-ends
- New IPC mechanism

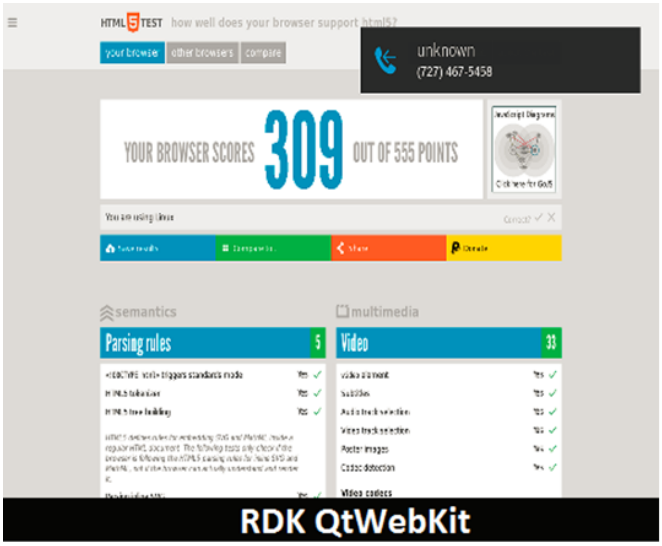


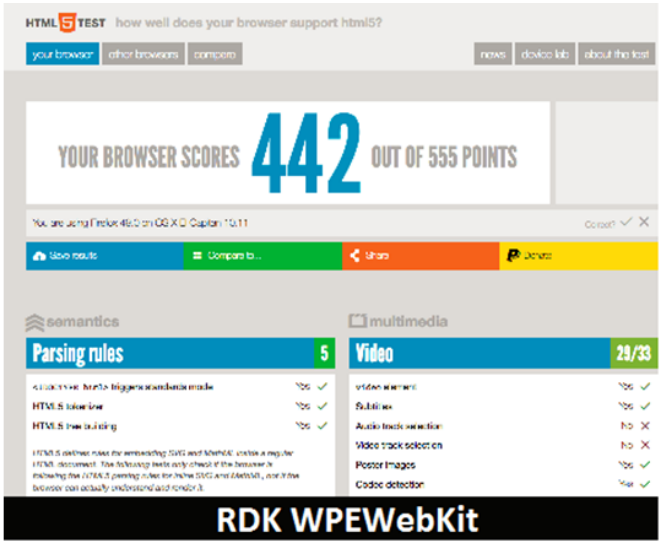
Work Accomplished in RDK

- Ported on RPI
- MIPS support for BCM targets
- Addition of Intel-CE and other BCM configurations
- MSE and EME implementation for YouTube leanback
- Parity or better than QtWebKit.
- Shadow Dom
- Wayland Support
- Fullscreen API
- MIPS JIT Architecture Change
- Walyland Reference Backend
- Speech APIs & reference implementation
- Geolocation
- Notifications
- PlayReady DRM
- Automated test runner
- IndexdDB Support
- Injected Bundles
- Full MSE Compliance
- WebCrypto
- View Backends for non-Wayland
- Gamepad Support

Functional Test (HTML5): QTWebkit Vs WPEWebKit

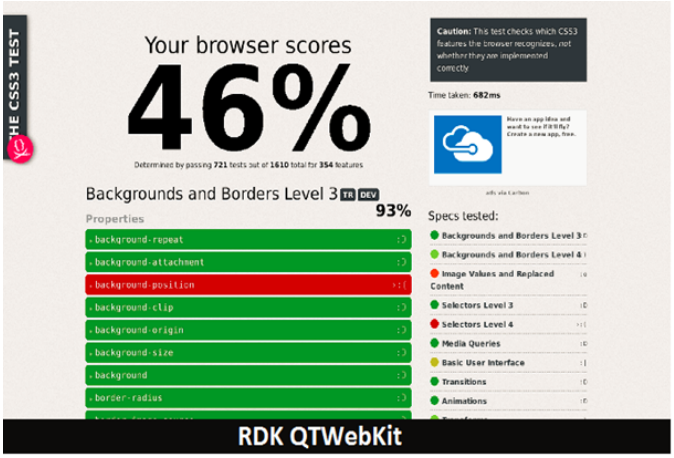
These tests are based on the site https://html5test.com/

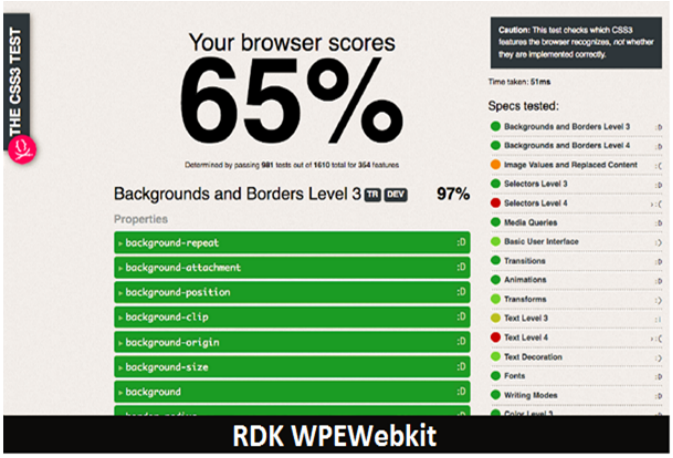




Functional Test(CSS3): QTWebkit Vs WPEWebKit

These tests are based on the site http://css3test.com/





Repositories

- WPE: https://github.com/Metrological/WebKitForWayland
- OE recipe: https://github.com/Metrological/meta-metrological
- Buildroot: https://github.com/Metrological/buildroot-wpe

WPE Support with RDK - Recipes

Weston:

- meta-rdk-ext/recipes-graphics/wayland/

  waylandwayland_1.6.0.bbwayland-native_1.6.0.bbwestonweston_1.6.0.bb

Westeros:

- meta-rdk-video/recipes-graphics/westeros

  westeros.bb westeros.inc westeros-simplebuffer.bbwesteros-simpleshell.bbwesteros-sink.bb

WPE Webkit:

- meta-rdk-ext/recipes-extended/wpe-webkit/

  wpe-webkit_0.1.bbwpe-webkit_0.2.bb wpe-webkit.inc

WPE Support with RDK - Build

- WPE Support is provided with specific image types
- Source meta-cmf/setup-environment
- Select the required machine configuration
- Build the image with WPE package support. E.g.
  - bitbakerdk-generic-hybrid-wpe-image
  - bitbakerdk-generic-mediaclient-wpe-image

How to work with WPE

- Step 1: Fork WPE on GitHub.com
- Step 2: Apply changes on your fork
- Step 3: Test your changes

- Step 4: Submit pull request to WPE master
- Step 5: Prepare for comments, rework if necessary
- Step 6: Wait for merge notification