Automotive Grade Linux compositor architecture

This document outlines the architecture and design for the Automotive Grade Linux reference Wayland compositor.

This specification is still under development, and should not be considered final.

¶ Background

¶ Client architecture

¶ Connecting to Wayland server

The Wayland server shall be able on the default `wayland-0` socket. Clients may connect to this socket and use the standard Wayland protocol with no additional requirements.

¶ Compositor architecture

¶ Overall functional components

libweston backend: output management and display

libweston renderer: complex composition

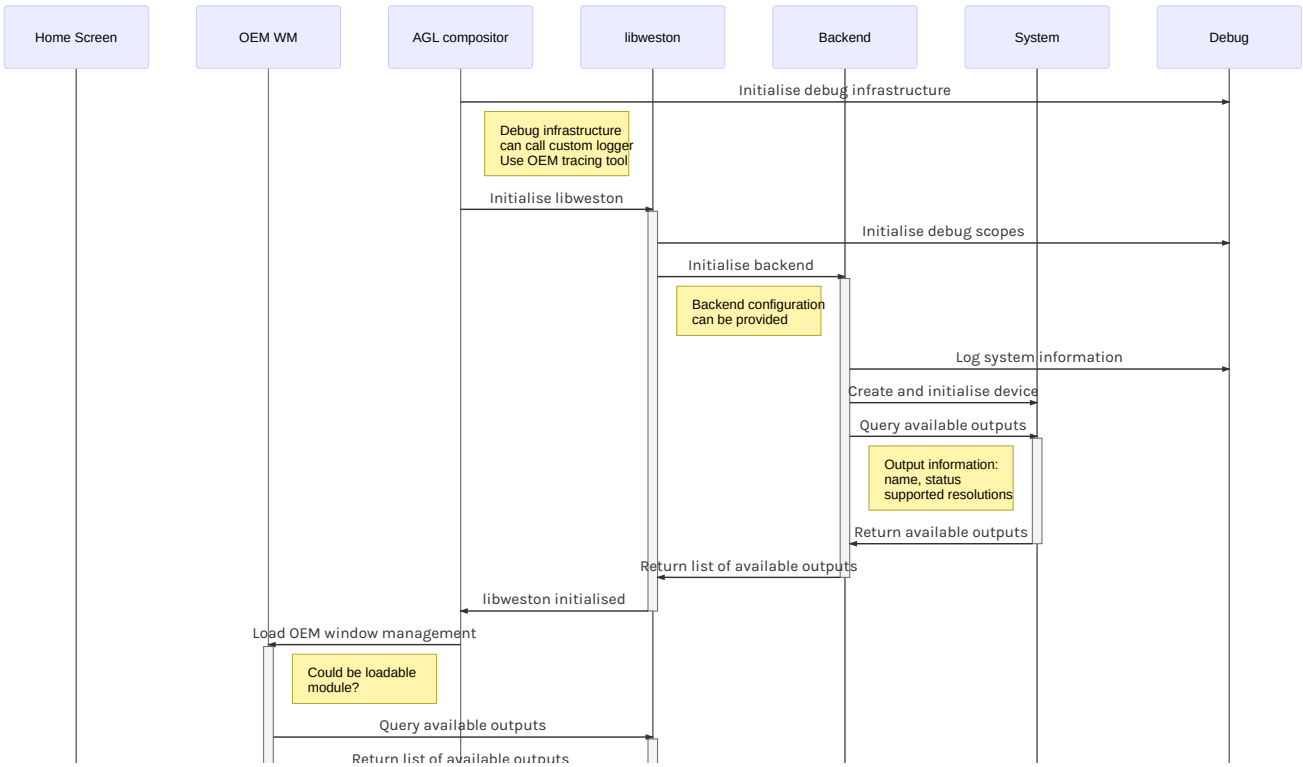
libweston core: accounting and window list

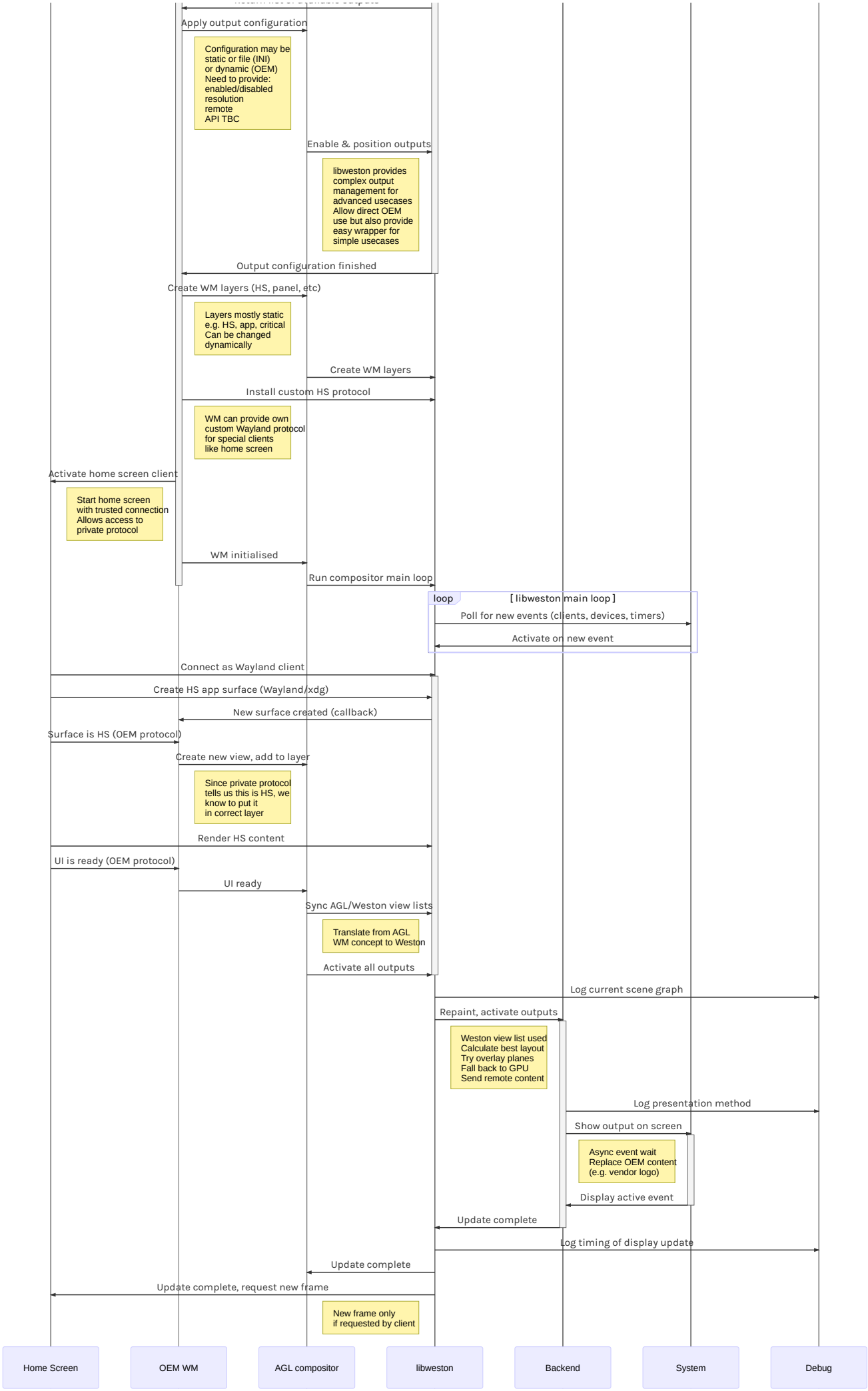
AGL compositor: configuration handling and helper libraries

OEM window management: WM policy

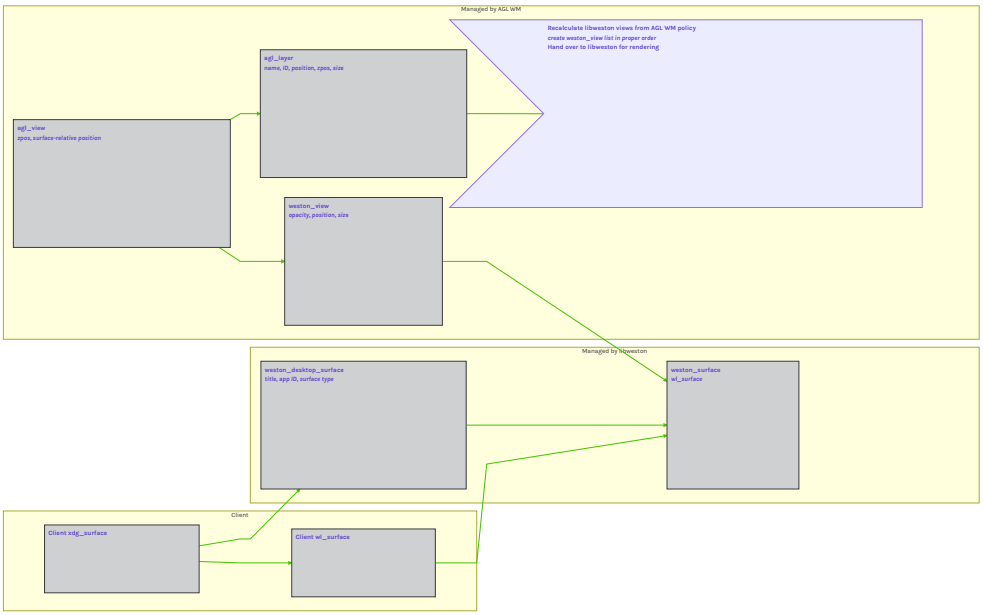
Home screen: Special functional UI (w special protocol)

¶ Initialisation


Home Screen
OEM WM
AGL compositor
libweston
Backend
System
Debug
Initialise debug infrastructure
Debug infrastructure can call custom logger Use OEM tracing tool
Initialise libweston
Initialise debug scopes
Initialise backend
Backend configuration can be provided
Log system information
Create and initialise device
Query available outputs
Output information: name, status supported resolutions
Return available outputs
Return list of available outputs
libweston initialised
Load OEM window management
Could be loadable module?
Query available outputs
Return list of available outputs


Apply output configuration
Configuration may be static or file (INI) or dynamic (OEM) Need to provide: enabled/disabled resolution remote API TBC
Enable & position outputs
libweston provides complex output management for advanced usecases Allow direct OEM use but also provide easy wrapper for simple usecases
Output configuration finished
Create WM layers (HS, panel, etc)
Layers mostly static e.g. HS, app, critical Can be changed dynamically
Create WM layers
Install custom HS protocol
WM can provide own custom Wayland protocol for special clients like home screen
Activate home screen client
Start home screen with trusted connection Allows access to private protocol
WM initialised
Run compositor main loop
loop
[libweston main loop]
Poll for new events (clients, devices, timers)
Activate on new event
Connect as Wayland client
Create HS app surface (Wayland/xdg)
New surface created (callback)
Surface is HS (OEM protocol)
Create new view, add to layer
Since private protocol tells us this is HS, we know to put it in correct layer
Render HS content
UI is ready (OEM protocol)
UI ready
Sync AGL/Weston view lists
Translate from AGL WM concept to Weston
Activate all outputs
Log current scene graph
Repaint, activate outputs
Weston view list used Calculate best layout Try overlay planes Fall back to GPU Send remote content
Log presentation method
Show output on screen
Async event wait Replace OEM content (e.g. vendor logo)
Display active event
Update complete
Log timing of display update
Update complete
Update complete, request new frame
New frame only if requested by client
Home Screen
OEM WM
AGL compositor
libweston
Backend
System
Debug

¶ Window managment



¶ Work plan

1. Prepare JIRA tickets and changes to spec
    1. Should this be developed as a parallel new spec, or evolution of existing specs?
    2. Auxiliary porting documentation to show clients how to port from old to new frameworks
    3. Where should architecture document live? Perhaps alongside code, as living document: track current status of codebase
    4. Review 1.0 HMI spec in Confluence, including usecases
2. Create AGL reference compositor repository
    1. Prepare generated documentation to be integrated with overall AGL documentation site
    2. Initial code from Weston repository
    3. Prepare for synchronization with new libweston features from upstream
3. Create AGL CIAT tests
    1. Ensure reference compositor is also buildable on top of AGL UCB
    2. Ensure reference compositor runs and successfully displays on reference hardware
4. Split reference window-management design into two models
    1. In-process model shall be preferred and model for initial development

    2. Support example plugin using `ivi_wm` protocol interface for external window management
5. Port reference client homescreen UI to new architecture
    1. Develop private Wayland extension for supporting homescreen as external client (similar to desktop-shell)
    2. Start and control homescreen client process from Weston
    3. Bring into compositor repository, or leave external?
6. ~~Develop 'context' API to support context-aware policy~~
    1. ~~Is the car stopped, moving, in gear?~~
    2. ~~Geolocation~~
    3. ~~Can we source this information from Signal Composer?~~
7. Create list of window management usecases and provide example implementations
    1. Push surface to front (switch application)
    2. Animate between split-screen and full-screen usecase
8. Review support for gesture recognition and bindings
    1. Should support swipe-to-switch-application usecase
    2. Ensure no content is shown before all homescreen content is ready
9. Implement extension to `libwayland-server` or `libweston` API to query SMACK label of client
    1. Submit support upstream for release
10. Implement configurable tracing control
11. Create work plan for multiple hardware backends
    1. Port old cluster demo to multi-backend work
12. Develop `agl_foreign_subcompositor_v1` extension for nesting surfaces between multiple processes, and prove this support in a real application
    1. Do we have a sample application we can use for this, or someone who can volunteer to port their own application?
13. Investigate and develop integration with CAN signaling input
    1. Define list of inputs required to forward
    2. Does this need to be per-vendor / per-device?
    3. Mainloop integration with reference compositor
14. Investigate surface-criticality extension
    1. Need to query app framework for criticality role: what is the API to do this?
    2. Need to review, merge, and extend explicit fencing support such that low-priority clients do not pre-empt critical clients
15. Select text-input extension to use and implement reference on-screen keyboard within compositor

1. Qt has a capable OSK we could re-use, but may need fixing and/or updates

¶ Follow-up work

1. Roadmap for application framework lifecycle work
    1. Follow app FW lifecycle discussion (is this scheduled?)
    2. Feed surface visible/active status back into app FW
2. Roadmap for notification proposal
    1. Review Dominig's presentation from Yokohama F2F about notifications
    2. Much more complex than just pop-ups, e.g. notifications from stopped applications, application badges (message app icon with number of unread messages), status banners (album art for currently-playing music)
3. Surface criticality extension
    1. What does critical content mean? Is it just window management - force window to front - or other?
    2. Align vocabulary and permissions with audio framework.
    3. For window management, 'priority' better wording than 'emergency' or criticality; usecases for CarPlay / Android Auto which should prevent any other content from displaying above it.

¶ Functional and design requirements

The following have been identified as functional requirements which must be fulfilled by the reference AGL compositor.

1. OEM customisable and replaceable HMI/UI
    1. The home screen design shall allow the OEM to either customise the provided HMI user interface, or replace the provided interface with their own, using any available client toolkit.
    2. The window manager design shall allow the OEM to either customise the provided window management policy, or replace the provided policy implementation with their own.
    3. The window manager design shall allow the implementation to query the client SMACK label, allowing security policy to be enforced for window management.
    4. The reference HMI implementation shall provide for both fullscreen and split-screen application usecases, with top and bottom bars for system UI panels.
    5. The reference HMI implementation shall provide for a visual list of applications, to allow users to show a list of applications and switch between them.
2. OEM customisable input actions

    1. The implementation shall be able to capture particular key presses (e.g. 'radio' button) and take action based on these presses: either internal actions, or routing the presses to specific applications (e.g. route play/pause button to media player).
    2. The implementation shall be able to recognize particular touch gestures (e.g. swipe/zoom) and take action based on these gestures (e.g. swipe to switch active application).
3. Window management API shall be based on layers and views
    1. Similar to existing `ivi_wm` and `libweston` API: views are ordered and grouped within layers, layers are ordered and grouped on outputs
    2. Integer-based helper API shall be provided for easier management
4. Display server and window management API must not assume display size or orientation
    1. Display server must natively support different display sizes, and allow display size to be configured.
    2. Display server must support different orientation (portrait/landscape) and allow orientation to be configured.
5. ~~Server applies policies from changing context (situation, drive engage, country, ...)~~
    1. ~~The server shall make it possible for OEM modules to subscribe to events from CAN or other (e.g. geolocation) signals, allowing changes to be made when context changes (e.g. hide media playback when gear is engaged).~~
    2. ~~The OEM modules shall be able to prohibit particular applications, or classes of applications, from running at any time, and to forcibly switch applications or show notifications.~~
    3. The above requirements should be handled by OEMs, or activity manager, or some other component/EG.
6. Assurance that critical content is presented rapidly
    1. Content declared as critical shall be presented immediately, and not be blocked by non-critical clients.
7. Server preserves previous content until new content is ready
    1. If OEM content (logo) is being shown, when display server starts, this content shall not be replaced until the full user interface is ready to be shown.
    2. As in below requirement, when the full user interface is ready to be shown it shall be presented immediately and in full.
    3. As an exception, when the reverse camera is being shown, the HMI shall not be shown until reverse camera is disengaged.
8. Every transition frame correctly shown
    1. During animations and transitions, each frame shown shall be a correct intermediate step for the given time in the animation.

    2. Frames showing incomplete or incoherent content shall not be shown to user.
9. Reference compositor can run without support for X11 clients or dependency on X11
    1. The reference compositor must not have any dependency to another window system. It must be possible to run AGL in an environment where only Wayland is supported.
10. Nested Wayland and X11 backends (local development usecase)
    1. Optionally, these 'nested' backends shall be provided for development and testing purposes.
11. DRM/KMS backend (native display usecase)
    1. The reference backend shall use kernel modesetting (KMS) and the atomic modesetting API for display control.
    2. This backend shall use hardware overlay planes for composition where possible, to reduce power requirements from GPU and increase overall image quality and reliability.
12. Support for vendor-provided display backends (virtualization or multi-ECU)
    1. Vendors with their own display backends shall be able to implement separate display backends, e.g. for display in a virtualized or multi-ECU usecase.
13. Support for EGL / OpenGL ES composition inside display server
    1. For advanced effects or when overlay planes are not usable, the server shall use EGL and OpenGL ES 2.0 for final composition.
    2. This will be implemented using the GBM API to use EGL on top of KMS.
14. Support for vendor-provided rendering backends inside display server (2D compositor IP block)
    1. Many automotive platforms support 2D composition using a dedicated IP block and interfaces such as V4L2. The reference server shall make it possible to implement a separate renderer.
    2. This renderer should be designed with thought to virtualized usecases.
15. Support for software-only composition inside display server (hardware bring-up / testing)
    1. Software-rendered clients shall be supported at all times, even when display server is hardware rendered.
16. Support for input from common vehicle sources
    1. Direct touchscreen input, taken from `libinput` and `evdev` devices
    2. CAN inputs such as buttons, knobs, wheels, sourced from AGL Signal Composer

3. On-screen keyboard to provide text input from touchscreens

17. App framework signal/event forwarding (CAN/voice user input)

    1. The compositor shall provide forwarding of CAN inputs which needs to be routed to the currently-active client, or according to window management policy, e.g. steering wheel yes/no buttons, scroll wheel, previous/next track.
    2. The compositor shall *not* provide forwarding of CAN inputs which do not be routed according to any policy, e.g. steering wheel orientation, current temperature, accelerator.

18. Support for standardised Wayland extensions

    1. `wl_output` and `xdg_output_v1` shall be present to describe display devices
    2. `xdg_wm_base` shall be present for window management; clients should not use the deprecated `ivi-application` interface
    3. `wl_subcompositor` shall be present to support combination of multiple surfaces into a single presentable entity
    4. `zxdg_importer_v1`, `zxdg_exporter_v1`, `agl_foreign_subcompositor_v1`, shall be present to support combination of multiple surfaces from different client processes
    5. `wl_shm` shall be supported for software-rendered clients, with at least `WL_SHM_FORMAT_XRGB8888` and `WL_SHM_FORMAT_ARGB8888`
    6. `zwp_linux_dmabuf_v1` should be present to support zero-copy presentation of hardware-generated content from clients (media/GPU)
    7. `wp_presentation` shall be present to support presentation timing feedback to clients
    8. `wp_viewporter` shall be present to support cropping and scaling of content
    9. `wl_seat` shall be present for keyboard, button, wheel, and touch input
        1. `wl_keyboard` shall provide key and button events
        2. `wl_touch` shall provide direct touchscreen events
        3. `wl_pointer` shall provide wheel and knob events
    10. **TBC**: One of the `text_input` family of extensions shall be present to provide text input from an on-screen keyboard

19. Inter-process 'foreign' surface relationships

    1. Clients shall be able to compose their user interface from multiple processes.
    2. The `wl_subcompositor` interface allows clients to compose a user interface from multiple surfaces, e.g. media and UI surfaces which are layered and tethered to each other. However, these surfaces must be from the same client connection.
    3. The `zxdg_importer_v1` and `zxdg_exporter_v1` interfaces allow clients to transfer handles to surfaces between different processes,

however they only allow usage as XDG shell dialogs.

4. An `agl_foreign_subcompositor_v1` extension shall be developed which allows usage of the `wl_subcompositor` protocol with foreign surfaces. This extension shall be made generally available.
5. Example usecases: media player with media playback and UI inside separate processes.
6. Rich notification content: applications may embed notifications and content (e.g. media album art) inside notification bar controlled by HMI / System UI.

20. Support for EGL / OpenGL ES clients using standard Wayland EGL platform

    1. Clients using `libwayland-egl` and `EGL_KHR_platform_wayland` shall be able to display content.

21. Client request tracing available through AGL monitoring

    1. The architecture document shall define a set of trace sources which will be available through standard AGL logging, monitoring, and tracing, frameworks. These sources shall be customizable at runtime.

¶ Assumptions

1. The Wayland server is a critical system process, and must be reliable

    - If the Wayland server crashes, no new content can be presented. A wait to restart the server may result in a delay showing new content which would violate safety constraints. Therefore, it is not acceptable for the server to crash in regular use, and a crash or loss of responsiveness shall constitute a system failure event.

2. Window management constitutes a critical component of the Wayland server

    - The window manager is responsible for policy decisions determining the final presentation and display of on-screen content. If the window management module is unresponsive, content from existing clients may continue to be presented, however new surfaces or clients will not be presented. As per the above requirement, this constitutes a critical failure as it may violate safety constraints requiring prompt presentation of certain types of content.

¶ Discussion: May 2019 F2F

1. Do we have our OEM usecases?

    - Can we create some?
    - Integration with cluster IC content; multi-ECU punchout

    - Ohiwa-san has some usecases listed in the 2.0 HMI spec on Confluence
2. What are our critical outcomes?
    - Need to list these this week.
3. What are the most critical areas to document and illustrate?
    - Window manager policy: what is the workflow from application creating a surface, to how the WM places the content in a particular spot?
    - Full-screen to split-screen usecase
    - 'Cut out' areas for system UI
    - Placement of notification
    - HiDPI support
    - XDG `configure` event: how does the client know what size it should be, and what orientation it should render in? Finish up sequence diagram, and how does this integrate with IVI shell?
    - XDG `active` and focus events: app framework lifecycle management should understand app status - do we need to push information to AFW?
    - Which parts of `xdg-shell` can applications rely on to function? Fullscreen, move/resize, etc, denied by server.
    - How to expose new protocol to clients?
4. Timelines: when can we make the cut over to replace the HS/WM API?
5. Is there a particular configuration-file format which we should be using?
    - Would be nice to have an `include` directive, or a `config.d` directory to compose multiple files together.
6. Do we need special handling for hardware paths for backup camera?
    - Assume we may be running on a hypervisor?
7. Configure screen regions for system UI
    - Take diagram from HMI spec?
    - Support changing size of split-screen regions
8. Do we need to port the cluster demo to multi/dynamic-backend API?
    - Resolved: yes. This is part of the CES demo.
    - How should we communicate from the application to the server that the surface should be displayed remotely?
9. How do we communicate surface criticality from application to server?
    - Do we need a new extension to allow privileged clients to declare a surface is critical?
    - Requirement for app framework: role to allow client to present critical content.
    - Document requirement for single active critical surface at one time.
10. Do we need to support changing orientation dynamically?

- Resolved: no. We do not (currently) need to dynamically rotate the display.

11. What client protocol do we expose to handle popups and notifications?
    - Including changeable banners, e.g. currently-playing track and album art.
    - Review Dominig's presentation on notifications from Yokohama.
    - Create sequence diagram / decision tree for popup-display policy.
    - How do we deal with notifications, e.g. from inactive apps, launching those apps, docking to systray?
12. Do we need support for ivi-application clients using the IVI ID agent?
    - Proposed resolution: no. These clients are deployed in the field, however the proposed changes to window management and home screen will already require a change in these clients. This means that they can change to use the XDG shell at the same time.
13. Need to change demo Qt apps for resolution/orientation independence
    - Some simple apps can be made independent by using dynamic layout
    - Other components (e.g. homescreen, dashboard) will need separate layouts for portrait/landscape
    - Would need to have done for landscape CES demo
14. Do we need to research input event delivery restrictions?
    - We may need to implement restrictions for trusted-UI applications which restrict input delivery. Do we need a framework for this? Prior art already from cluster displays.

¶ Document revision history

Version	Date	Author	Notes
0.9	10th July, 2019	Daniel Stone	Updated for ALS F2F
0.3	8th May, 2019	Daniel Stone	Updated from Wednesday F2F discussion
0.2	7th May, 2019	Daniel Stone	Updated from Tuesday F2F discussion
0.1	6th May, 2019	Daniel Stone	Initial revision of compositor architecture document