A project report on

GAMEPAD SUPPORT, TOUCH LOCK SUPPORT, and VTG IN webOS TV

Submitted in partial fulfillment of the requirements for the degree of

Integrated Master of Technology

In

Artificial Intelligence

by

MAYANK SAMADHIYA

19MIM10077

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
VIT BHOPAL UNIVERSITY
Bhopal – Indore Highway, Kotrikalan,

Madhya Pradesh - 466114, India

APRIL 2024

GAMEPAD SUPPORT, TOUCH LOCK SUPPORT, and VTG IN webOS TV

Submitted in partial fulfillment of the requirements for the degree of

Integrated Master of Technology

In

Artificial Intelligence

by

MAYANK SAMADHIYA

19MIM10077

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
VIT BHOPAL UNIVERSITY
Bhopal – Indore Highway, Kotrikalan,

Madhya Pradesh - 466114, India

APRIL 2024

Declaration

I hereby declare that the thesis entitled **GAMEPAD SUPPORT**, **TOUCH LOCK SUPPORT and VTG IN webOS TV** submitted by **Mayank Samadhiya (Reg. No. 19MIM10077)** to the School of Computing Science and Engineering, VIT Bhopal University for the award of **Integrated Master of Technology in Artificial Intelligence** is a record of bonafide work carried out by me under the supervision of **Dr Shweta Saxena**.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or University.

Signature: मयंक समाधिया

Name of the candidate: Mayank Samadhiya

Register Number: 19MIM10077

Data: 03/05/2024

Place: Bhopal

Internship completion certificate

ABSTRACT

The growing popularity of cloud gaming has made it necessary to build cutting-edge capabilities for webOS TV. We address this trend with an endeavor to expand webOS TV's control capabilities through the seamless integration of many gaming input devices. Even though webOS TV has developed to accommodate a wide range of gaming functions, one present drawback is that game controllers, such as USB gamepads, cannot be used to operate webOS TV. The project's objective is to close this gap by adding extensive support for a broad range of gaming input devices. By doing this project, we aim to enhance the entire user experience and provide a more customizable, immersive, and controlled interface for webOS TV gamers.

In modern television technology, the integration of touch-sensitive controls has become increasingly prevalent, offering enhanced user interaction and functionality. The touch lock feature aims to address common usability challenges associated with touch-sensitive interfaces on televisions. By implementing this feature, television manufacturers can enhance user experience and usability, particularly in scenarios involving remote control interactions or direct touch commands on screen. The touch lock functionality prevents accidental inputs on touch-sensitive screens, improving user experience by avoiding unintended commands such as channel changes or volume adjustments. In summary, touch lock support in televisions optimizes usability, reduces user frustration, and elevates the overall viewing experience by ensuring deliberate user interactions.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. Shweta Saxena, Assistant Professor Senior Grade 2, SCSE, VIT Bhopal University, for his constant guidance, continual encouragement, understanding more than all, he taught me patience in my endeavor. My association with him / her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Computer Science.

I would like to express my gratitude to Dr. G.Viswanathan, Mr. Sankar Viswanathan, Ms. Kadhambari S. Viswanathan, Dr. Senthil Kumar Arumugam, and Dr. S. Poonkuntran, School of Computer Science and Engineering, for providing with an environment to work in and for his inspiration during the tenure of the course.

In a jubilant mood I express ingeniously my whole-hearted thanks to Dr. S.Devaraju, Program Chair and Senior Assistant Professor SCSE, all teaching staff and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Bhopal

मयंक समाधिया

Date: 23/01/2024 Signature of the Candidate

Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

1	QML	Qt Modelling Language
2	RCU	Remote Control Unit
3	MRCU	Magic Remote Control Unit
4	FD	File Descriptor
5	LSM	Luna Surface Manager
6	webOS OSE	webOS open-source edition

Organization Information

LGSOFT is a software development company known for its expertise in creating solutions for various industries, including telecommunications, finance, healthcare, and more. LGSOFT specializes in developing custom software solutions tailored to the specific needs of its clients.

Its areas of expertise include Telecommunications software, Finance Software, Healthcare software, and other industries.

LGSOFT employs a team of highly skilled software engineers, developers, and technology experts proficient in a variety of programming languages, frameworks, and technologies. This includes but is not limited to:

- Programming languages: Java, C#, Python, JavaScript, etc.
- Development frameworks: .NET, Spring, Django, Angular, React, etc.
- Databases: SQL Server, Oracle, MySQL, PostgreSQL, MongoDB, etc.
- Cloud platforms: AWS, Microsoft Azure, Google Cloud Platform, etc.
- Mobile development: iOS, Android, cross-platform frameworks like React Native and Flutter.

LG Corporation is a multinational conglomerate corporation headquartered in South Korea, which is best known for its electronics division and production of televisions. LG Electronics, a subsidiary of LG Corporation, is a major player in the global consumer electronics and home appliances industry.

LG TVs are often equipped with smart features like webOS, which is the company's proprietary platform for smart TVs. This platform gives users access to various streaming services, apps, and other online content. LG has also been investing in AI-powered features for their TVs, making the overall user experience more convenient and enjoyable.

LG has a strong presence in India and has become a reliable brand. The company operates in various sectors within the country, providing a wide range of consumer electronics and home appliances. The Indian market has access to LG televisions, refrigerators, washing machines, air conditioners, and smartphones, demonstrating the corporation's commitment to meeting the diverse needs of consumers.

Learning Objectives/Internship Objectives

- The main objective is to offer interns practical experience in fixing bugs, developing software, and executing projects.
- To provide interns with technical expertise in QT/QML annotations and libraries.
- To introduce interns to cutting-edge technologies, devolving and bug-fixing knowledge, with a focus on utilizing C++, Qt, and QML.
- To provide interns with the opportunity to work with industry experts and professionals in the technical field.
- Learning about C++, Linux, and Bux fixing technologies
- To foster teamwork, collaboration, and communication skills among interns.
- To enable interns to apply their theoretical knowledge in a practical setting and gain practical experience in problem-solving, bug-fixing and critical thinking.
- To help interns develop their professional network by interacting with peers, mentors, and professionals in the industry.

CHAPTER 1
INTRODUCTION

1.1 Introduction

The increasing popularity of cloud gaming has spurred the development of new features in webOS TV. To provide a more engaging and flexible gaming experience, our team is working to address a significant limitation – "**The Inability to use different game input devices, like USB or Wireless gamepads, to control webOS TV.**"

webOS TV has made significant progress in accommodating various gaming features. However, it currently lacks compatibility with game input devices, which limits users from fully utilizing their gaming peripherals. This report proposes a strategic initiative to seamlessly integrate a wide range of game input devices into the webOS TV ecosystem.

The main goal of this initiative is to provide webOS TV users with more control and flexibility by allowing them to use their preferred game input devices to navigate and interact with the TV interface. We aim to overcome the current limitations and enhance the overall user experience, making webOS TV an even more attractive platform for gaming enthusiasts. This report outlines the reasons behind our efforts, our objectives, and the expected outcomes of empowering users with comprehensive game input device support on webOS TV.

1.2 Motivation of the work

Our motivation is rooted in the desire to provide users with a seamless and immersive gaming experience, where the choice of input devices becomes a crucial element of their interaction with webOS TV. As more and more people use webOS TV as a versatile gaming platform, we understand the importance of improving the system's control capabilities. The purpose of this effort is to connect the changing gaming world with the features of webOS TV, creating a more captivating, adaptable, and user-focused entertainment platform. We aim not only to meet the present-day expectations of gaming enthusiasts but also to anticipate and adjust to their future requirements through this initiative.

1.3 Why GAMEPAD Support is Required

- The number of users playing cloud games is increasing day by day, and webOS TV has been updated with new features to support various input devices. However, there is one drawback that needs to be addressed – "**users are currently unable to control webOS TV using gamepad controllers.**"
- Currently, when playing cloud games on webOS TV, users have to navigate both the TV RCU/MRCU and a gamepad controller, which results in a challenging user experience.
- With this support, Users can easily control webOS TV and Cloud Gaming with gamepad controllers.

CHAPTER 2

RESOURCE REQUIRED AND BUSINESS VALUES

2.1 Resource Required (Software / Hardware)

- **Mobaxterm:** Used for connecting to TV and Servers using IP's
- **Teraterm:** Used for connecting to a TV using a serial cable
- **Cloud PC:** a cloud-based development environment.
- **Wall server:** Similar to GitHub, this storage server is designed to store and keep track of code changes.
- **Polar Server:** Data storage server
- **Notepad++:** An Integrated Development Environment (IDE) application, used for developing and debugging code.
- **VS Code:** Another IDE platform for code changes and debugging.
- **QT / C++:** Used as the backend programming language.
- **QML:** Used as the frontend programming language.
- **Gamepad:** Hardware
- **USB:** For connecting wired Gamepads

2.2 Customer Values and Business Case

- We aim to improve the usability and accessibility of LG webOS TV for our new business area and cloud gaming services.
- As a cloud gaming service requires external devices, we aim to expand and provide a convenient environment by supporting a variety of devices for our users.

2.3 Possible Usage

2.3.1 Gaming Experience

Enabling gamepad support enables users to use their favorite gaming controllers to play a variety of games. This can include casual games, more immersive gaming experiences, or even multiplayer games that benefit from the precision and responsiveness of game controllers.

2.3.2 App Navigation

Gamepads can be used to navigate through the webOS TV interface. This means that users can easily browse through menus, access different apps and settings, and perform various basic actions with the game controller. This feature is especially helpful for users who find using a remote control less intuitive.

2.3.3 Text Input

Gamepad can be used as a convenient tool for keyboard input. This is especially valuable when users need to search for content or enter login credentials, whereas a traditional remote control might be less efficient.

2.4 Summary

Gamepad functionality has been added to webOS TV, which provides a more flexible and improved user experience. Gamepads provide a simple way to manage the playing of media. Users can easily manage various media-related operations, change volume levels, and browse through streaming applications with ease. Additionally, gamepads can be used for more than just normal webOS TV interface navigation. They offer a convenient way for users to navigate menus, open multiple apps, and perform different tasks. This feature provides a distinct and convenient way to enter data as a text input device.

CHAPTER 3
FEATURE REQUIREMENT

3.1 Introduction

Gamepad support on webOS TV must allow smooth integration of various game input devices, whether they are wireless or USB-based. This feature should enable users to easily navigate the interface, control media, and enhance their gaming experience. This feature must be user-friendly, providing flexibility and accessibility for a diverse range of users.

3.2 Disadvantages/Limitations in the existing system

- The current system does not provide support for connecting different types of gaming input devices for cloud game users.
- While playing cloud games on a webOS TV, users face difficulty in using both the TV RCU/MRCU and the gamepad controller, leading to a poor user experience.

3.3 Proposed System

We are developing a gamepad support mechanism that is specifically designed for webOS TV keeping the user's needs in mind. This system can work with a wide range of wired and wireless game input devices, providing users with greater control and a user-friendly interface. Our primary objective is to enhance the gaming experience by optimizing interactions with different controllers, which will lead to increased engagement and immersion. We are trying to support a broad range of peripherals to ensure that our products are accessible to everyone. Our user interface ensures a seamless transition between gaming and other activities, offering a smooth user experience. Overall, our main aim is to make webOS TV more dynamic, adaptable, and user-friendly.

CHAPTER 4

SYSTEM DESIGN AND IMPLEMENTATION

To implement this **GAMEPAD SUPPORT IN webOS TV**, initially, we are trying to support given Gamepad devices.

- Sony Dual Shock 4
- Xbox 360 controller
- NVIDIA Shield
- Luna Amazon controller
- SHAKS controller

To support this Gamepad feature we mapped the Game keys to our webOS and below table is given for that information

Gamepad	**webOS**
Left Thumbstick or D-pad	4-way key navigation
Right Thumbstick	Up/Down: Volume Control
Right Thumbstick	Left/Right: Channel Control
A-button	Enter (confirm)
B-button	Return (cancellation)
Press and hold B-button	Finish
Menu or Option Button	Settings
Xbox or Home Button	GameHub (If supported) or Home
Press and hold the Xbox or Equivalent Button	During Cloud Gaming: Quick Panel

4.1 Key Mapping Information Diagram

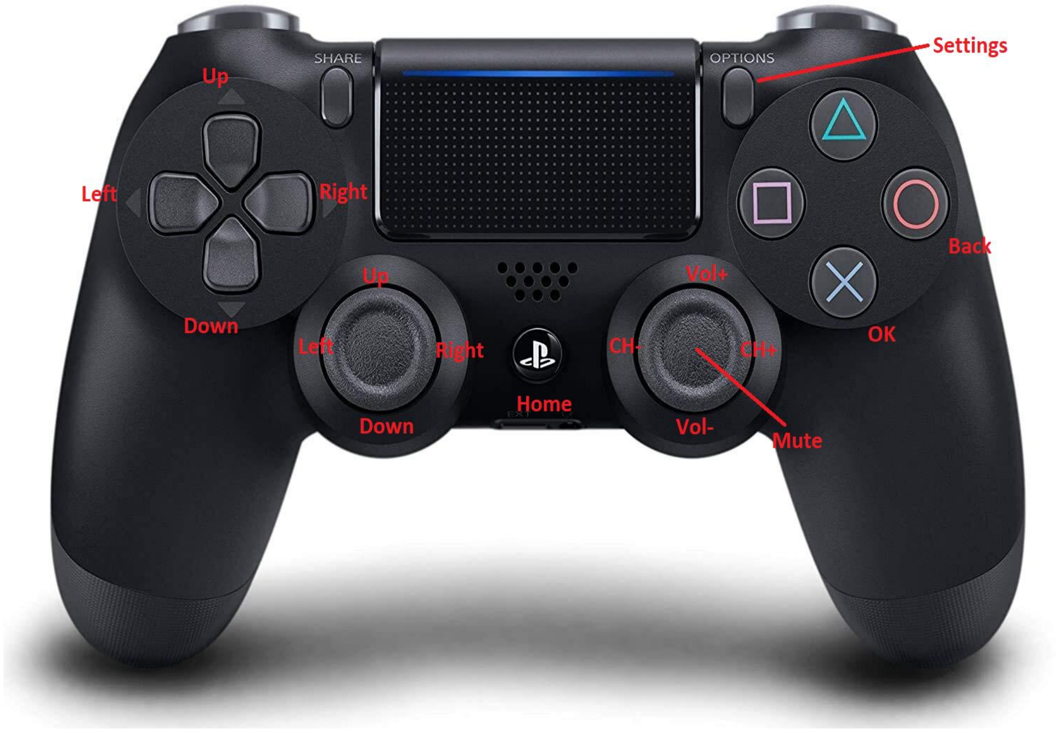


Figure 4.1.1 Sony PS4 controller

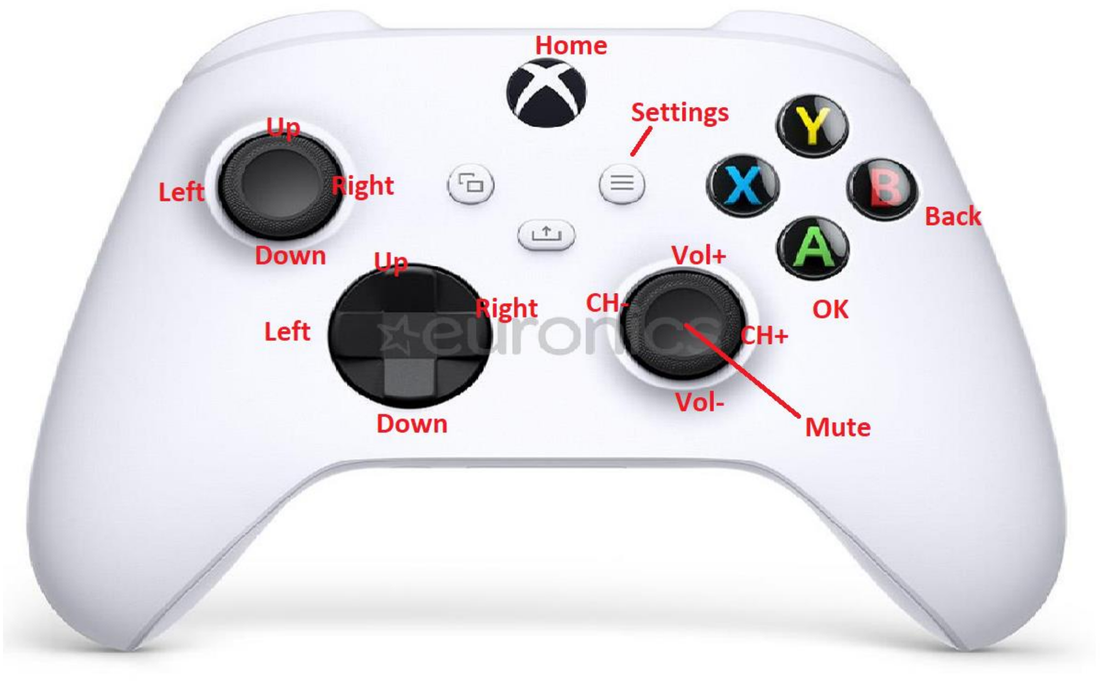


Figure 4.1.2 Xbox Controller

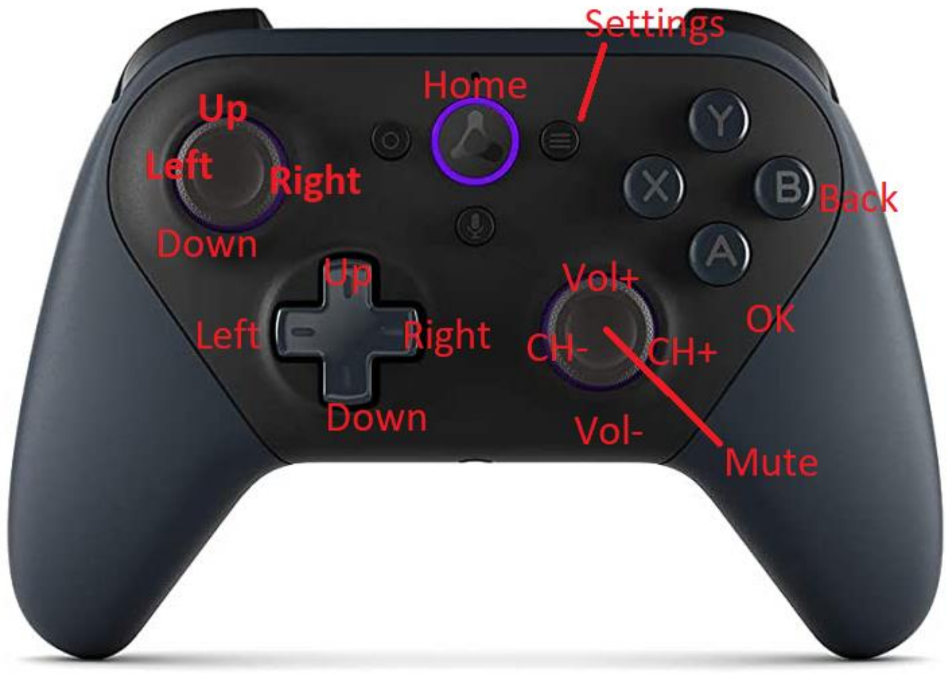


Figure 4.1.3 Amazon Luna Gamepad Controller

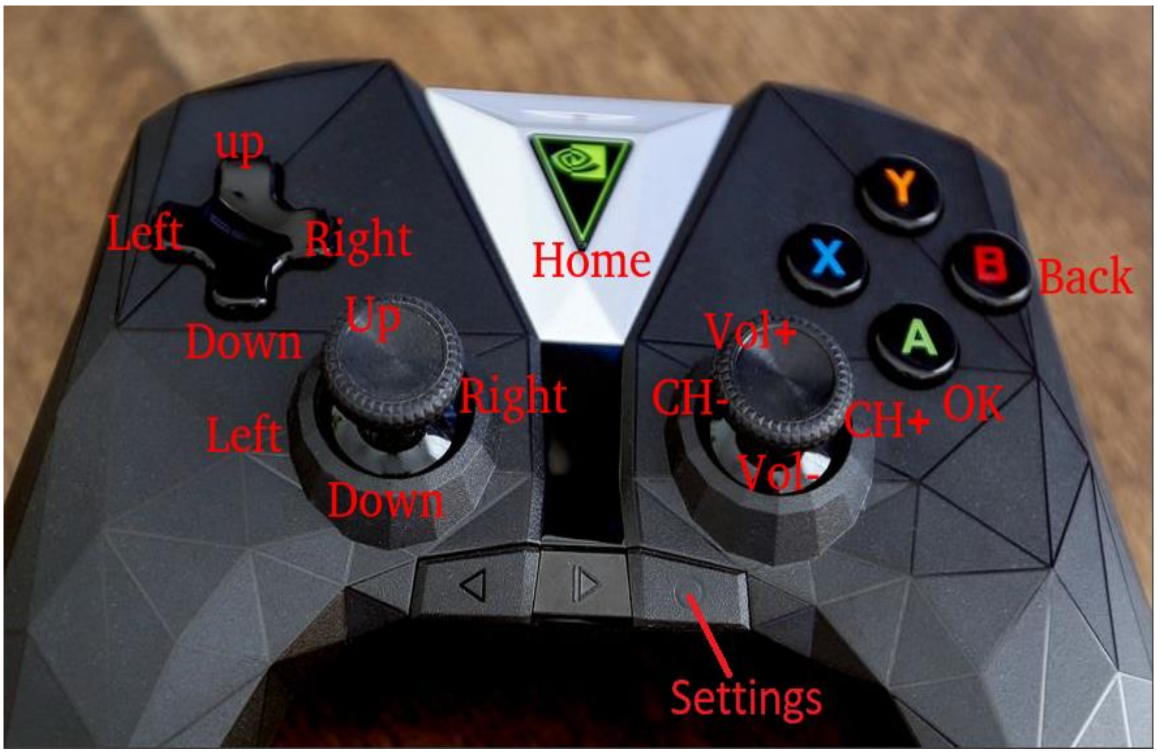


Figure 4.1.4 NVIDIA Gamepad Controller

4.2 How Key is getting Handled in between Different Modules

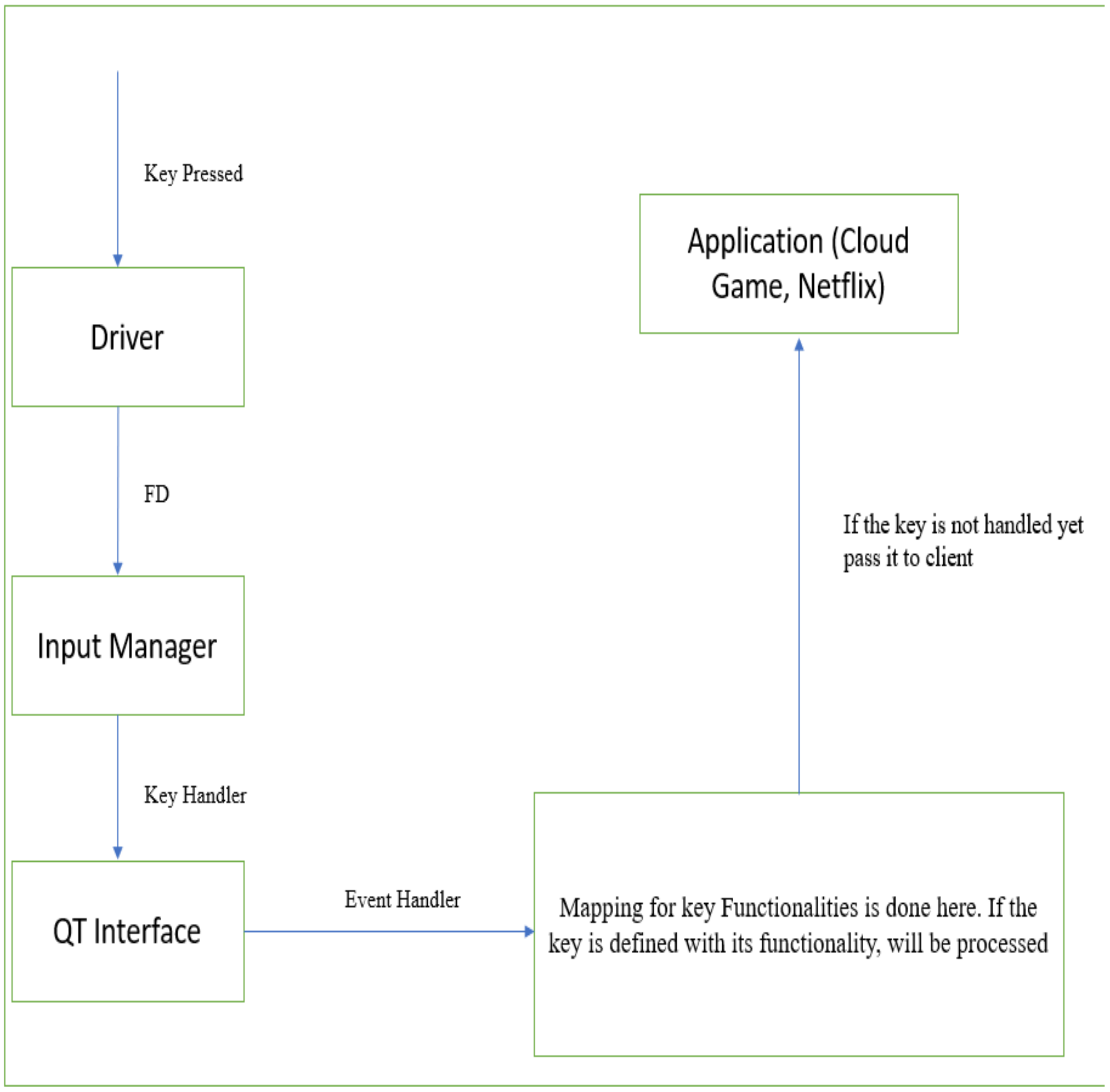

Key Pressed
Driver
FD
Input Manager
Key Handler
QT Interface
Event Handler
Mapping for key Functionalities is done here. If the key is defined with its functionality, will be processed
If the key is not handled yet pass it to client
Application (Cloud Game, Netflix)

Figure 4.2.1 Code Flow for Key Event

4.3 Challenges Faced till Now

- **Racing Condition** – Proper handshaking between Input Manager and Browser to avoid key Conflicts

➢ **Solution:**

1. Required separate property from browser team to identify cloud game is launched and cloud game gets closed to LSM.

2. Using that property will process/avoid Gamepad keys while the user is in the cloud game

- **Thumb stick navigation –** The input values from the thumbstick were axis values (-32768 to 32767) that need to be converted into a single key event value.

➢ **Solution:**

1. We added a logic that converts the input range value to a single navigation keycode.
2. Consider scenarios where the thumbstick fails to settle at the midpoint, but is moved again for navigation. In such cases, the navigation key code should be passed.

- **Repeat Functionality** – Not Received REPEAT event from the driver during Press and Hold any key in Gamepad Controller

➢ **Solution:**

1. Introduced a New QTimer (QT Class Object) to handle the REPEAT event for Gamepad controllers.
2. After receiving the Press event for any key, the Timer will start and if the timer reaches the Timeout and the Release event is not happening So timer will continuously generate and send the press event until the Release comes for the same key.

- **Unnecessary Thumb stick Event During Idle State –** Received Unnecessary key events from Some Gamepad controllers in Idle state

➢ **Solution:**

1. Added a Logic to avoid (Blocking) the same key during the Idle state

4.4 Other Activities

Worked on Different TV Issues and Feature Verifications

- **Issues**
    - Custom Cursor on Live TV
    - Last Input type for Key Input in TV
    - Cursor Position in Portrait Mode
    - Linux Crash Issue in TV after the Long run
    - Channel switching in Between CS1 and CS2 for Japan Region
    - Spinner and Alert are not being displayed at the same type
    - LG Channels video is not playing in App View when the focus is coming to LG Channels 2nd time
    - LG Channels key is not working for the India Region
    - While entering into Web Cloud games, webOS functionalities are working when VKB is launched for Gamepad Controllers

- **Testing**
    - **Touch Lock Screen** Feature Verification
    - Gamepad key Verification in Monitor / Tunerless models
    - AD/SAP key description for Smart monitors
    - Navigation between decoration buttons in the Multi-View controller
    - Racing condition in between webOS functionality and SDL/NDL Application functionalities
    - Double Cursor Issue in the browser application

CHAPTER 5

Touch Lock Feature Implementation

5.1 Requested Feature

- Add Touch Lock as a Settings Menu Item
- Place it in the <**Accessibility**> menu or <**Other settings**>
- Put a "Quick Action" button on the Quick Settings as an option so that the customer can add it if desired, <**Display Touch lock/unlock status**>, <**Apply on/off switch**>
- When the touch lock menu is selected, the execution intention is reconfirmed through a pop-up
- When the screen is touched in the touch lock state, the lock toast is displayed at the top right corner

5.2 Customer Values and Business Cases

- Parents can lock the touch screen to prevent accidental or unauthorized usage by children, ensuring a safer environment at home.
- Users can secure their privacy by locking the screen when stepping away from the TV, preventing unauthorized access or viewing.
- Incorporating touch lock technology can distinguish OUR TV models from competitors, attracting customers looking for enhanced safety and usability features.
- Offering advanced features that prioritize user needs can boost brand reputation and customer satisfaction, leading to increased brand loyalty and positive word-of-mouth.

5.3 Implementation Details

- Blocking The Touch event in Input Manager if the Touch Lock value is on
- The Touch Lock feature has been introduced to the Settings menu. This means that when the Touch Lock value is altered from the settings menu, the input manager will receive the updated information via an API call made in between modules. Events will be processed further based on the Touch Lock value that was received from the settings menu.
- If Touch Lock is on and touch is getting received then sending the toast notification to show the toast on to the screen.
- Unlocking the Touch Lock with the help of the local key sequence, added key sequence for power key, volume up, down, up, and down keys, when these volume keys are pressed from the stand by me TV the touch will be unlocked, and user will be able to use touch feature.

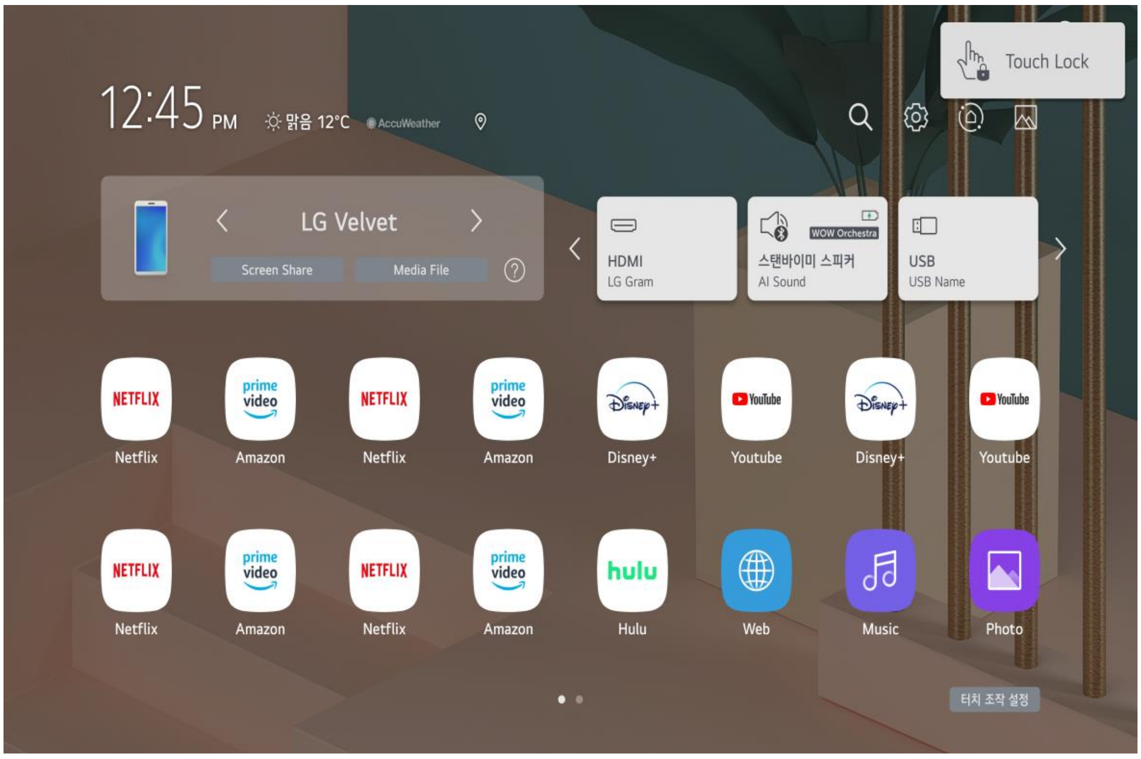


Figure 5.3.1 Screen Touch Lock

CHAPTER 6

Sign Language Zoom Feature

Sign Language, a means of communication through bodily movements, especially of hands and arms, is used when spoken communication is impossible or not desirable. The practice is probably older than speech. Sign language may be as coarsely expressed as mere grimaces, shrugs, or pointings; or it may employ a delicately nuanced combination of coded manual signals reinforced by facial expression and perhaps augmented by words spelled out in a manual alphabet. Wherever vocal communication is impossible, as between speakers of mutually unintelligible languages or when one or more would-be communicators are deaf, sign language can be used to bridge the gap.

To help such people who suffer from these problems and to keep them aware of what is going on in the world, to help them be entertained by the content expressed using sign language, so many television programs are providing this feature that they will deliver the content in both the mediums in audio as well as with the sign language. But the problem with this feature is that these windows which these service providers are providing are very small, which makes it less accessible or feasible for deaf people to understand the content.

To help with this problem, this feature will help efficiently by providing a special window to the user through which he/she can access the sign language window or any content of screen that the service provider is providing into a larger window and in two sizes which user can select as per his/her choice. User can enable or disable the feature as per their requirement by pressing the red key available on the LG-specific remote.

Our Project is to resolve this problem we have implemented a feature, in which we are providing an option for the user by which he/she can enlarge any particular area of displayed content television by creating a small new window at the bottom right corner of the display.

6.1 Why Do We Need it

webOS is a software platform that is designed to be web-centric and user-friendly for smart devices. It has demonstrated its performance and stability in over 70 million LG

- You can easily check sign language on a larger screen and watch TV.
- Improved convenience in setting automatic subtitles for non-sign language content
- To provide an optimal sign language screen for the hearing impaired while watching TV, provide a function to expand sign language sales using the multi-view / magic zoom function
- In the case of content provided simultaneously in sign language, the hearing-impaired provides multi-view and/or expansion of the sign language area through Magic Zoom
- Enhanced access to subtitle settings for content that does not provide sign language when watching TV for the hearing impaired
- Video capture service will provide the current video image and VtG library will provide the method to bind the video to graphic texture. The QML plugin shall provide the method to the QML layer that the app uses and shall provide the texture to the upper layer so that the application can resize and render.

6.2 Resources Required

- Application that deploys QML plugin to render video resize. This is an existing Live TV and Input app in LG TV.
- Target component to be developed; which shall provide the interface to the upper application and deploy the VtG library below.
- VtG Library development; Binds Live TV video and HDMI video to texture with the use of a library containing API in C++.
- Optimize the Input Common app due to VtG Plugin performance requirements if needed
- An option to Select the sign language zoom item in the Settings app to provide the Sign Language Zoom menu as Output.

6.3 Customer Values and Business Case

- Ap You can easily check sign language on a larger screen while watching TV
- Improved the convenience of setting the automatic subtitle function for non-sign language content

6.4 Possible Usage

- Can display any streaming video that can be captured by libVT(a module in TV that captures the content of the display and provides other services that require it), as a QML object.

6.5 Feature Requirements

To implement this Sign Language Zoom feature, we need three things to do: -

- Sign Language Zoom – Menu
- Sign Language Zoom – Select Area
- Sign Language Zoom – LiveTV

6.5.1 Sign Language Zoom - Menu

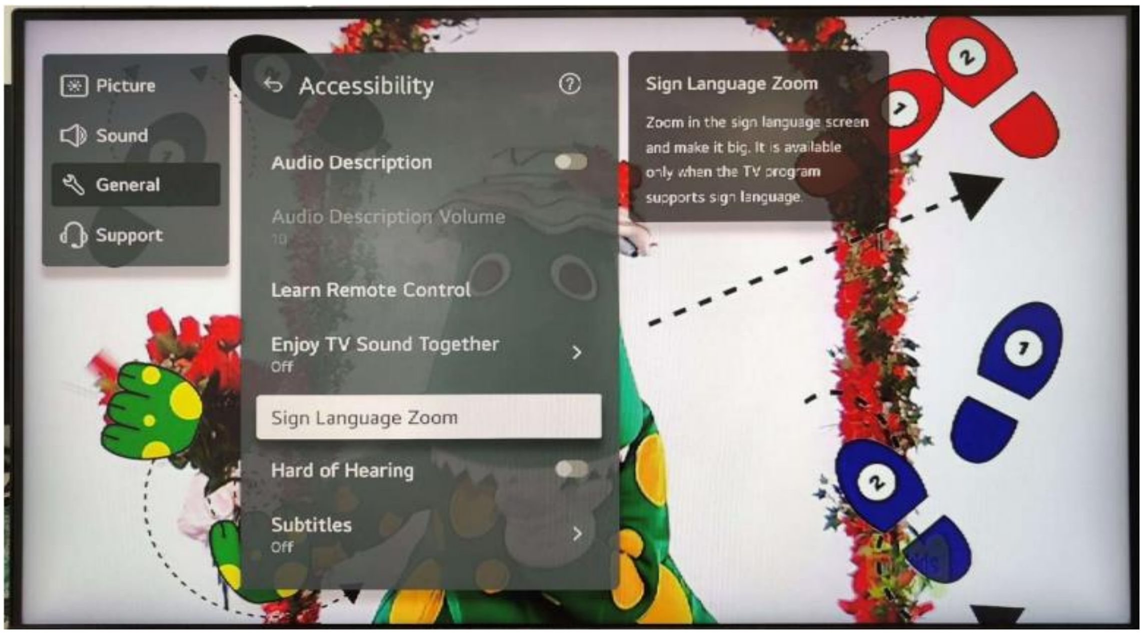
A menu on TV to provide user access to Sign Language Zoom from which he/she can activate or deactivate the Sign Language Zoom window to provide the content for that specific window which will be coming at the bottom right corner of the screen after activation.

To get the functionality of the sign language zoom feature, it needs to be enabled from the sign language menu, which will be available in the settings app below mentioned path: - Settings - General - Accessibility - Sign Language

The Input for Sign Language Zoom – menu will be to select the item in the settings app as the output of which it will provide a sign language zoom menu that supports the video or stream input from either DTV i.e., live feed or HDMI.

Item Name	Action when Selected
Activate	On/Off toggle
Select Area	Enter the Selection area mode screen

Table 6.5.1.1 The items to be displayed in the SLZ – menu



Sign Language Zoom

Zoom in the sign language screen and make it big. It is available only when the TV program supports sign language.

Fig. 6.5.1.1 Sign Language Zoom – menu option

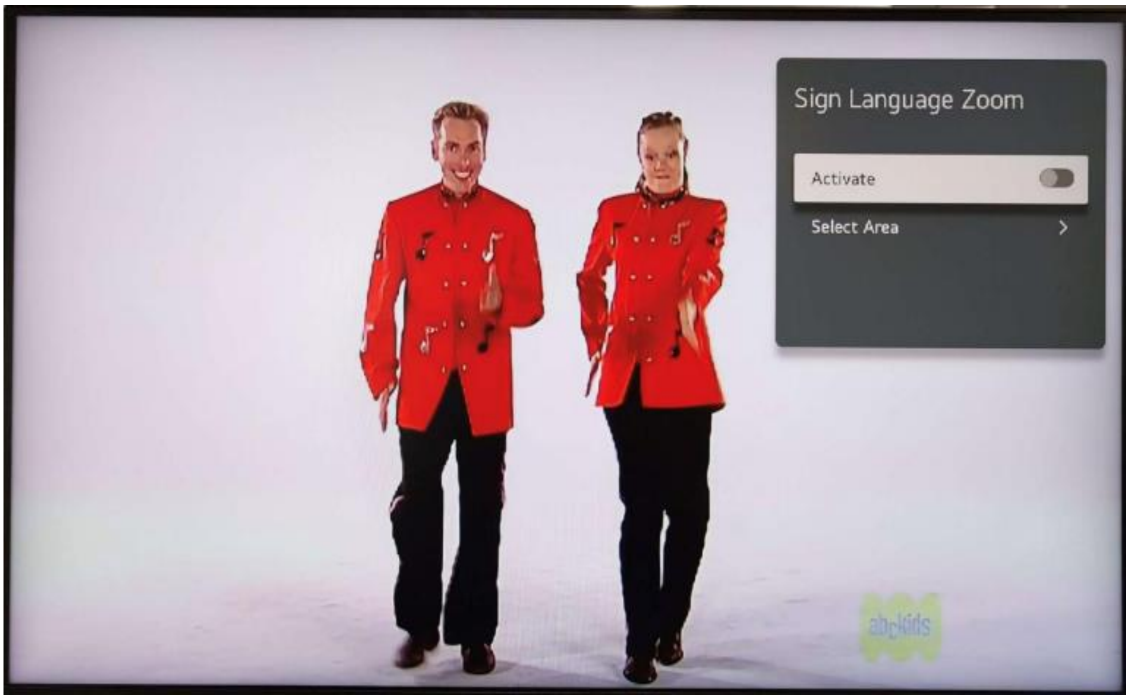


Fig. 6.5.1.2 Sign Language Zoom – menu

Activate the toggle button which is present in the menu to enable or disable sign language zoom is off by default whenever we are entering the LiveTV / HDMI app. While maintaining the sign language menu launched only a few keys will work for this menu

1) OK key: to select any item visible in any menu either settings or sign language zoom menu
2) Navigation Keys (Up, Down): to go through between different items present in the menu.
3) Back Key: to access the previous part of the menu.
4) Exit Key: to completely close the menu

These keys operate while maintaining the menu: Keys for selecting menus, back/exit keys for ending, and keys not handled by LiveTV/HDMI apps. Keys that do not work while maintaining the menu: All keys handled by the LiveTV/HDMI app except for the keys that work ex. INPUT / HOME / SETTINGS keys, etc., are keys that run the corresponding apps and belong to keys that the LIVETV app does not process. Keys operated after release: Same as before menu output (Sign language zoom) Same as before response

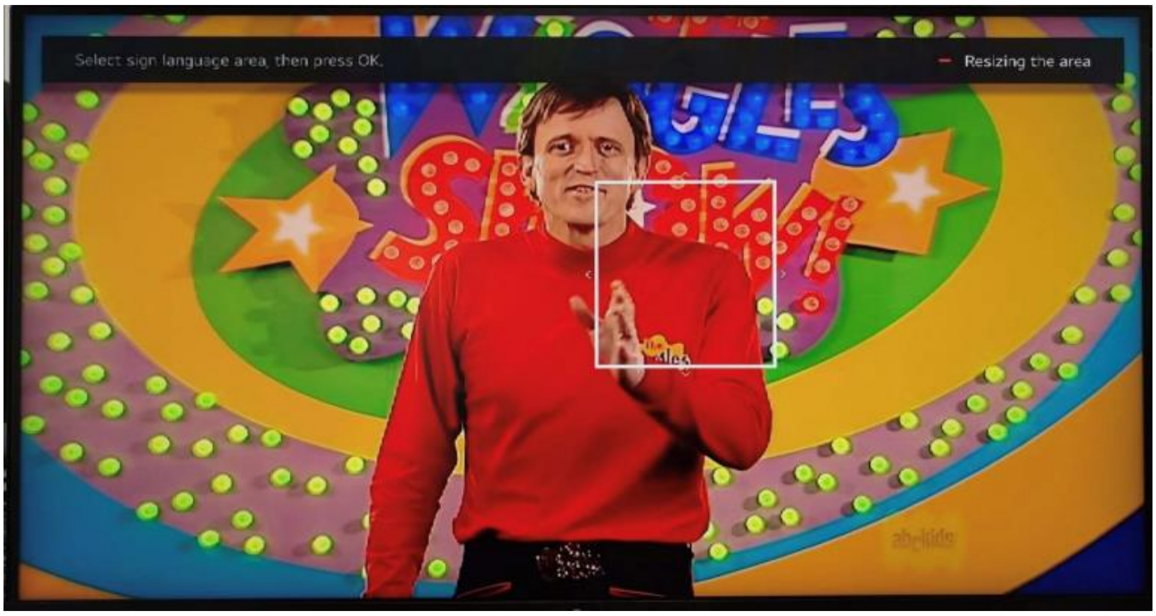
To determine whether a key is processed by the LiveTV/HDMI app, the external app (ex. browser, etc.) authorizes the key, and the keys supported by the function and the same operation as the key supported by the LiveTV/HDMI app are applicable. (ex. channel up/down, number keys, etc.). Since Live menu / EPG has a dash key function in a similar list in LiveTV/HDMI, it corresponds to the key processed by the LiveTV/HDMI app in the app, but it also works when the sign language enlargement menu is output because it works in an external app. (INFO key does not work because it is the key to output the default info pop-up within the app).

6.6 Sign Language Zoom – Select Area

To provide better accessibility for users, there is an option in the sign language zoom menu to give the user a feature to select the desired area of the screen to be displayed in the sign language zoom window. When the user selects this option of select area then it will enter the streaming window of TV either live TV or HDMI whichever will be in use at that time. In the select area interface users can move to any area using navigation keys (Up, Down, Left, Right) or by connecting to a mouse or magic remote (keeping the Audio guidance feature off) for easy accessibility, of the area and on pressing an OK key, streaming video of livetv/HDMI will be on screen with the selected area displaying in sign language zoom window.

For this part, the input will be the select area from the sign language zoom menu, which will provide the sign language zoom window as the output. When Select Area is selected from the sign language zoom menu, the Select Area screen is displayed.

On pressing the Red Color user can choose between two sizes of select area, like go to select area option livetv/ HDMI window will be available with select area rectangle, and the size of that rectangle can be changed between the sizes multiple times and by pressing the back or exit key users can access the previous window or completely exit from the feature and gets back to the livetv or HDMI.


Select sign language area, then press OK.

Resizing the area

Fig. 6.6.1 Select Area - Before Red Key Press

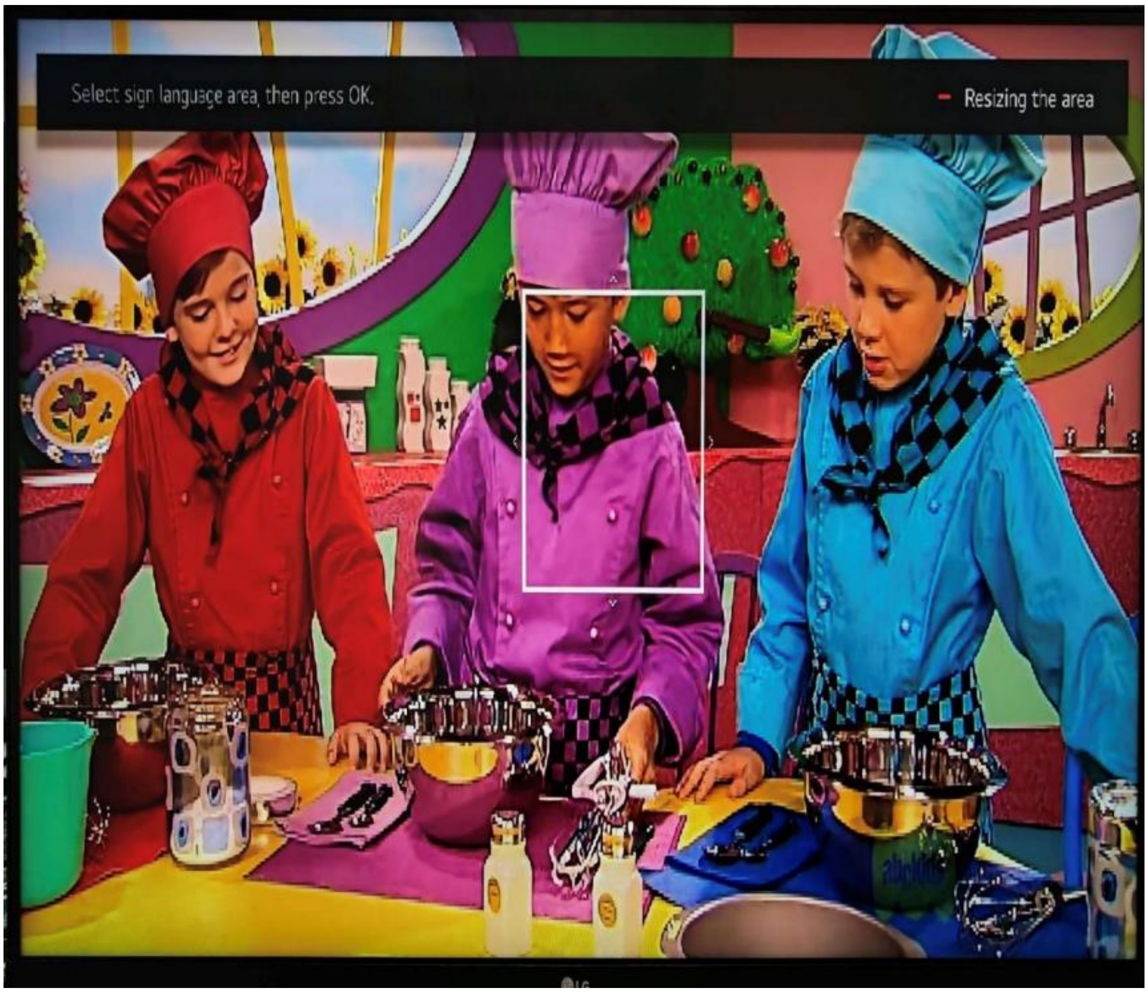


Fig. 6.6.2 Select Area - After Red Key Press

The Select Area screen is composed as follows.

1) Top: help text, color key help
2) Bottom right: Mark the area to be enlarged
3) Use the following keys for this function.

a. red color: 2nd step size toggle variable for the area to be enlarged
b. Directional keys: Adjust the position of the area to be enlarged
c. MRCU Cursor: The area to be enlarged according to the display conditions moves according to the cursor.
d. OK key: Run live zoom
e. back key: return to the previous menu
f. exit key: Exit all sign language select area/menu

Keys are handled independently of the LiveTV/HDMI app ex. Hotkeys to enter external apps: INPUT key, more action key, settings key, etc.

6.7 Sign Language Zoom – Select Area

As the output of the whole feature, the input for livetv will be Sign Language Zoom Enlarged area display execution, and for the output sign language zoom enlarged area execution. Using live TV comes with multiple scenarios to work with or to verify with. Working with sign language with live and HDMI depends in multiple ways like how we are switching between them, how we are activating and deactivating the feature

Scenarios:

- Live Zoom is displayed on the screen when activated or is set in the sign language enlargement menu or when selection is ended with the OK button in the select area.
- If you change activate off in the sign language enlargement menu, the display of Live Zoom is disabled.
- Activate on in sign language enlargement menu, activate off when switching input/app while Live Zoom is displayed, and Live Zoom is turned off.
- Activate on in sign language enlargement menu, activate off when power on/off in Live Zoom display state, and Live Zoom is canceled.
- Activate off in sign language enlargement menu, keep Live Zoom off when switching input/app in Live Zoom display off state.
- Activate off in sign language enlargement menu, keep Live Zoom off when power on/off in Live Zoom display off state.
- Live Zoom does not cover the subtitle/caption when displaying a subtitle in Live Zoom display mode. (subtitle/caption covers live zoom)
- If you enter the screen saver that cannot output video while Live Zoom is displayed, change the user setting to off.
- When entering a data broadcasting app (hbbtv, etc.) while Live Zoom is displayed, Live Zoom is not displayed on top of the data broadcasting app
- In the ATV channel or LG channel, active off is changed, and Live Zoom is canceled and not displayed.
- If you change back to the DTV channel, Live Zoom will not be displayed again.
- When teletext enters Live Zoom output state with active on, activation is changed to off, Live Zoom is canceled and is not displayed.
- If you end the teletext entry again, Live Zoom will not be displayed again.

- When multiview is entered, active off is changed, and Live Zoom is released and not displayed. (Both side by side / PIP)
- When entering multiview control mode in DTV & HDMI output state, active off is changed, and Live Zoom is canceled and not displayed.
- When multiview is closed, active off is changed, and Live Zoom is released and not displayed.
- When a video is displayed as a PIP partial screen instead of a full screen through data broadcasting and hybridtv functions, activate off is changed to cancel Live Zoom and not display. (When entering the menu, only activation is deactivated, and area selection remains activated)
- Sign language expansion is not supported in the delayed mode playback conditions of DVR and IP streaming playback conditions of data broadcasting. The sign language enlargement area is ended, the activation value is changed to off in the menu, and the activation process is displayed. However, the select area remains activated.

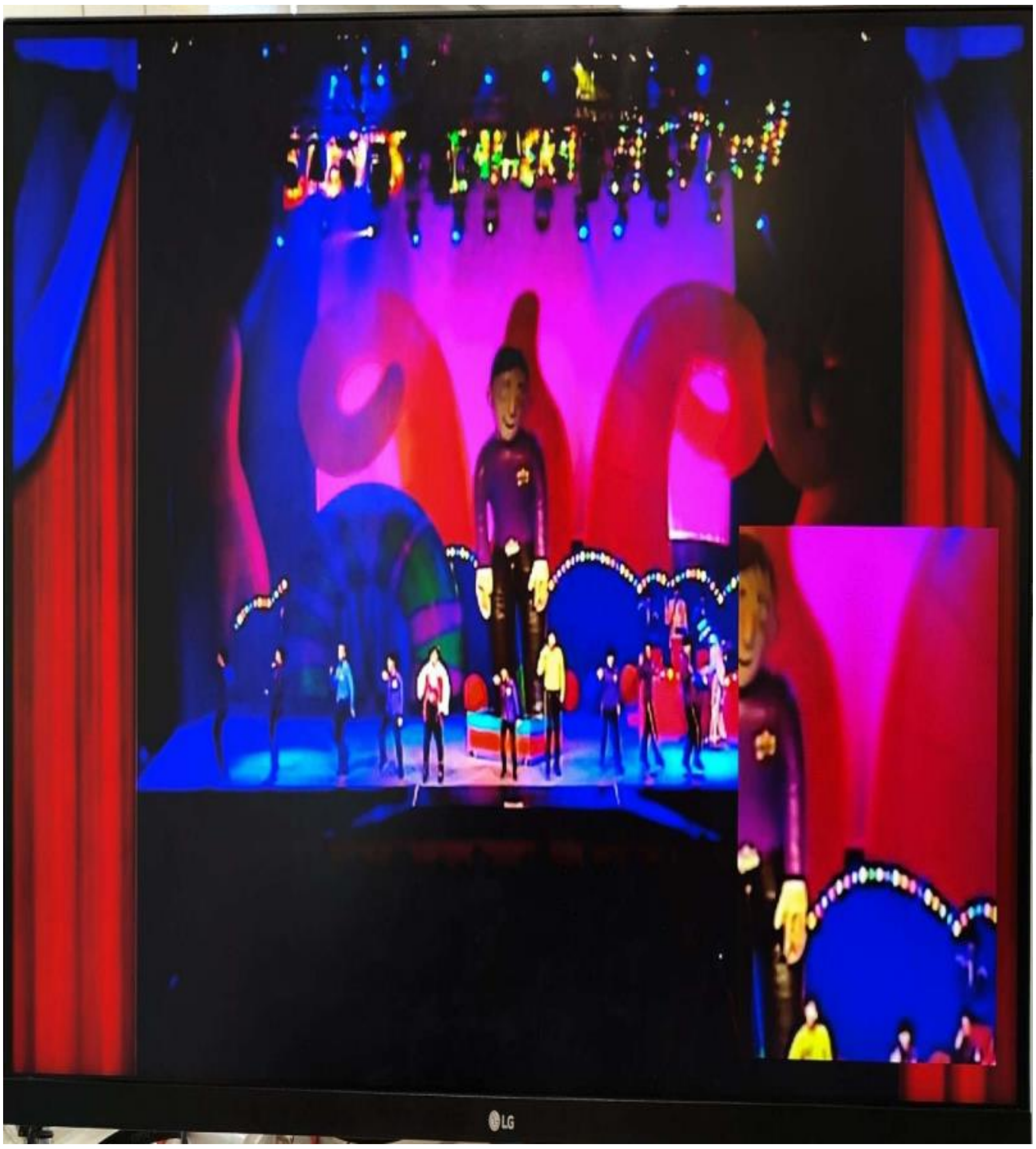


Fig. 6.7.1 Final Image of Sign Language Zoom

6.8 Sequence Diagram

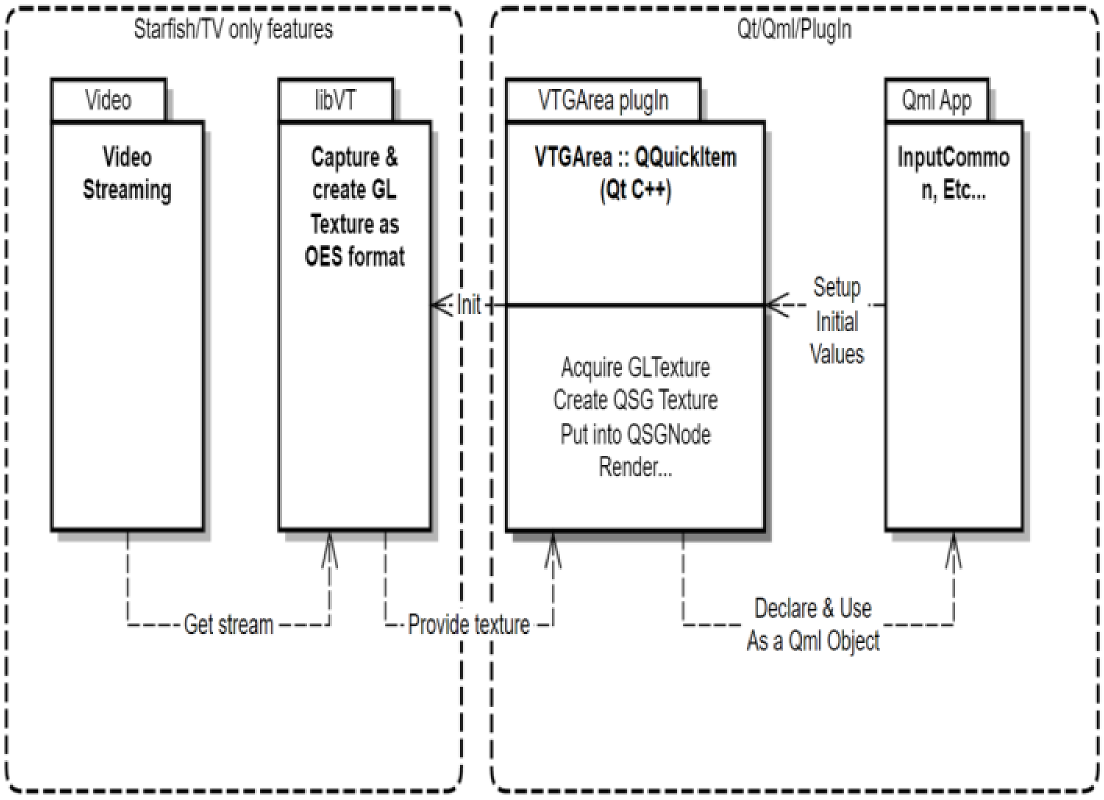


Fig. 6.8.1 Block Diagram for Sign Language Zoom

CHAPTER 7
webOS Architecture Overview

7.1 Introduction

webOS is a software platform that is designed to be web-centric and user-friendly for smart devices. It has demonstrated its performance and stability in over 70 million LG Smart TVs. Since its adoption for display products, webOS has evolved into a software platform that can be used for a wider range of products.

The open-source version of webOS is called webOS Open-source edition (OSE), and it was introduced in March 2018. It is based on the philosophy of open platforms, partnerships, and connectivity. webOS OSE offers additional features on top of the core architecture of webOS, which makes it suitable for use across a broad range of industry verticals.

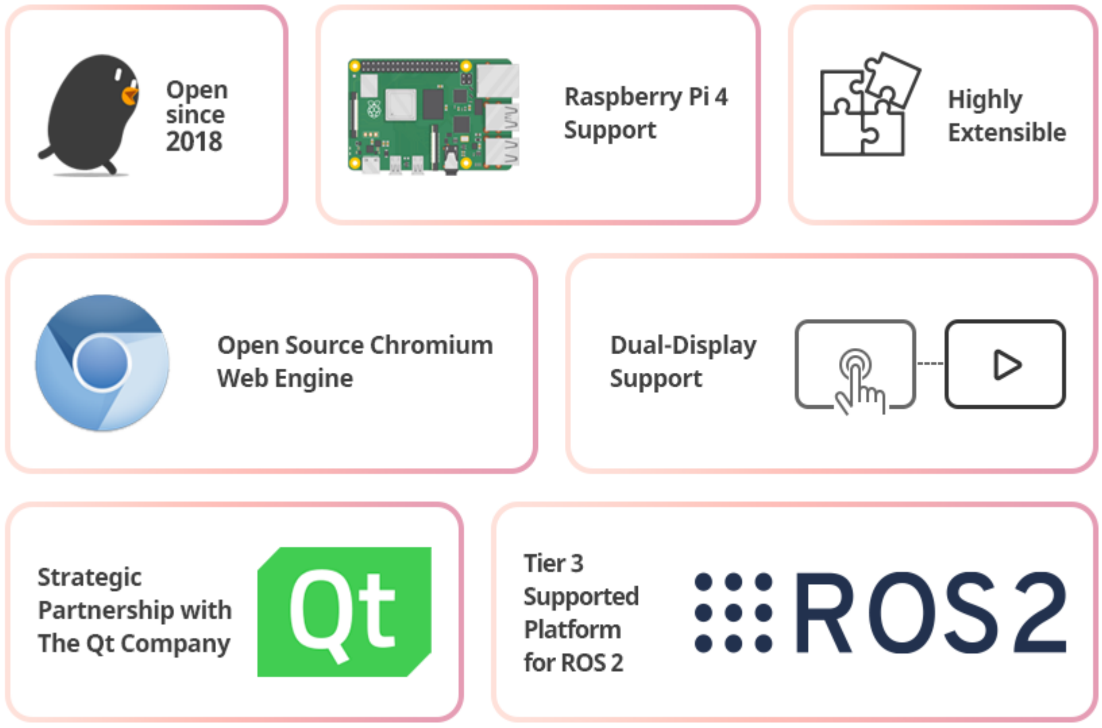

Open since 2018
Raspberry Pi 4 Support
Highly Extensible
Open Source Chromium Web Engine
Dual-Display Support
Strategic Partnership with The Qt Company
Qt
Tier 3 Supported Platform for ROS 2
ROS2

Figure 6.1.1 webOS OSE and it's wide reach

7.2 Interface Overview

webOS OSE's UI consists of two major UI components:

- App Bar
- Status Bar

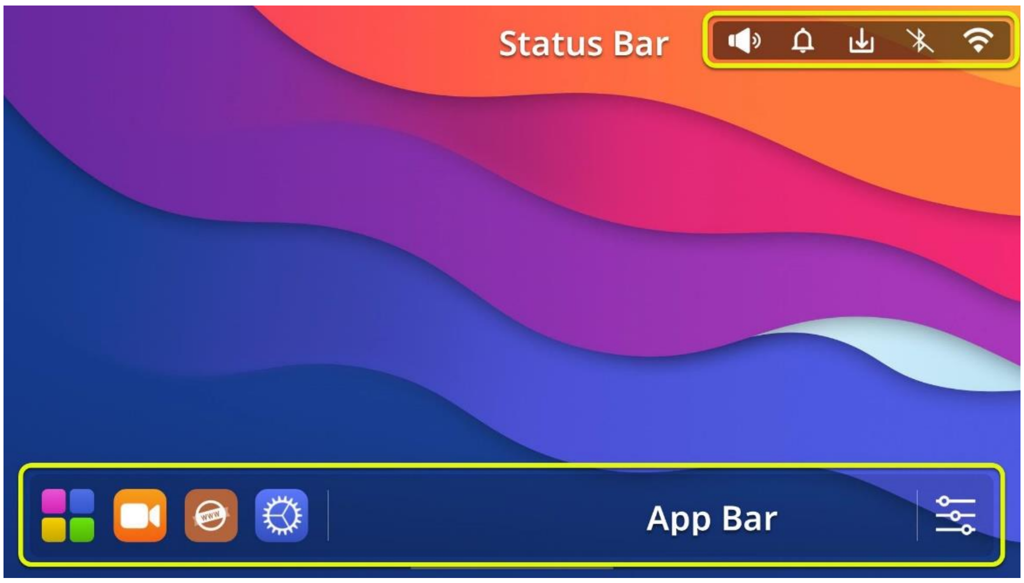


Figure 6.1.2 webOS OSE UI

7.2.1 App Bar

The app bar is the core interface of webOS OSE. This bar provides quick access to your favorite apps or currently running apps.

How to Access:

- Swipe up the bottom of the screen.
- Press the **Start** key on the keyboard.

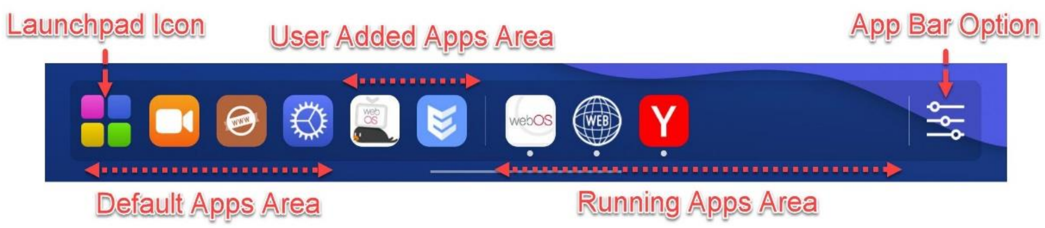


Figure 6.1.2.1 webOS OSE App Bar

7.2.2 Status Bar

The status bar is the shortcut for system settings, such as volume and notifications.

How to Access:

- Swipe down the top of the screen.

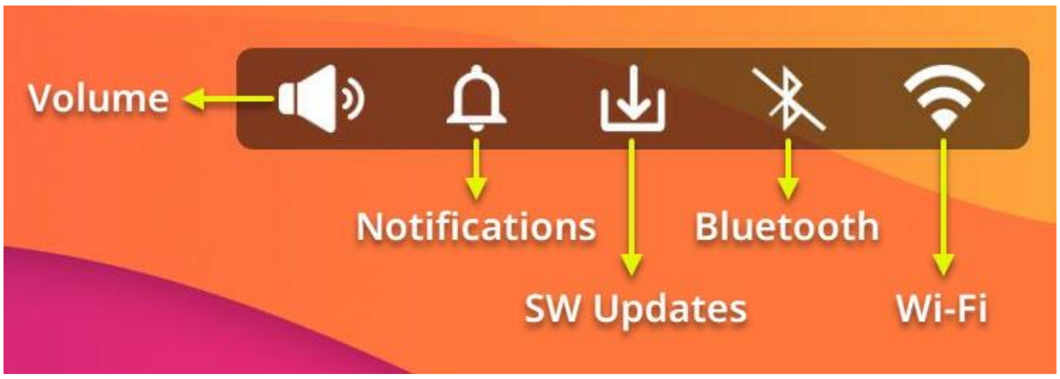


Figure 6.1.2.2 webOS OSE Status Bar

7.3 webOS Overview Architecture

webOS OSE is composed of different layers, including the Core Applications, Application Framework, Managers & Services, Base Components, and BSP/Kernel.

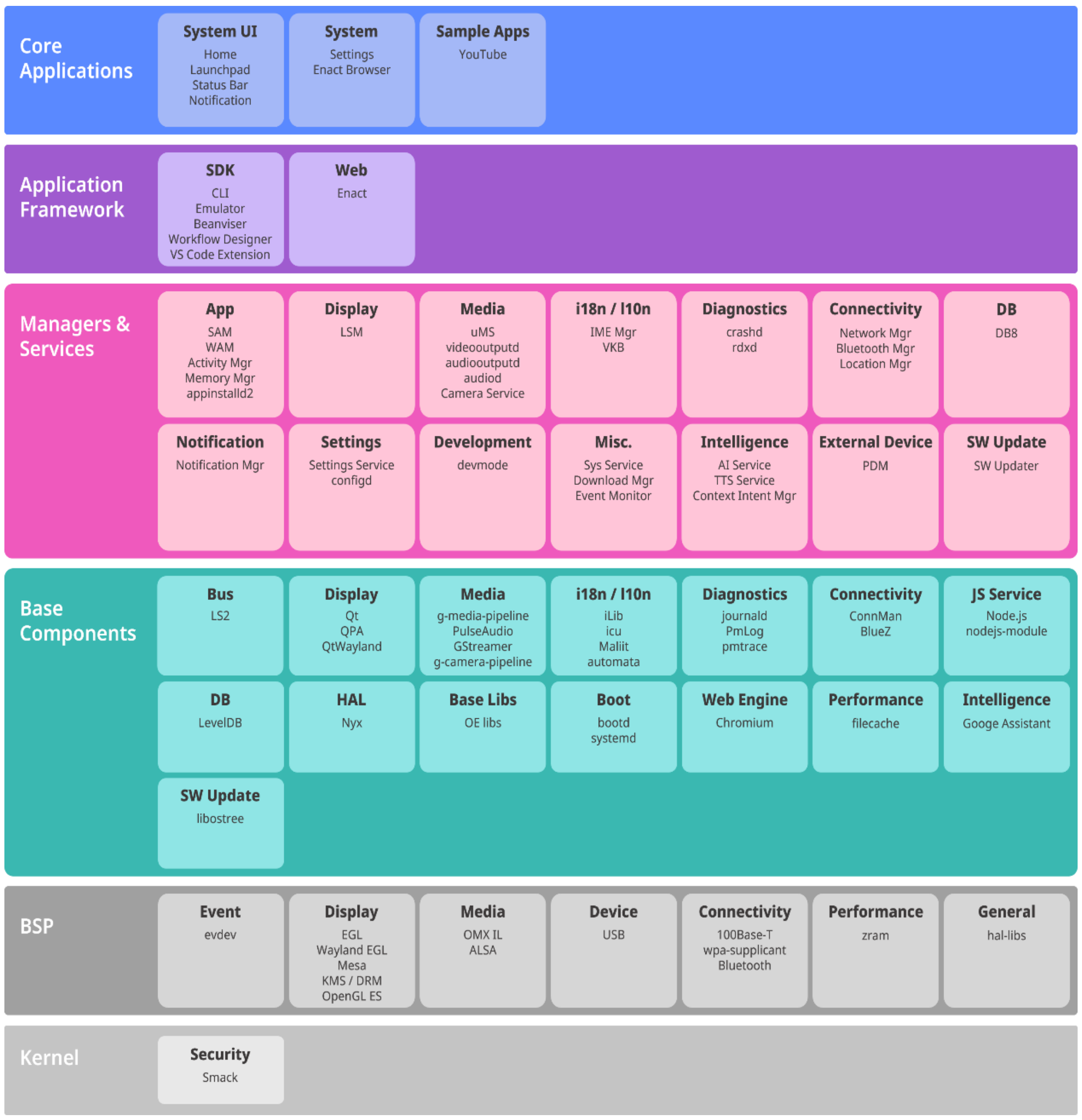


Figure 6.3.1 webOS OSE Architecture Overview

7.3.1 Core Applications

webOS OSE has Core applications as the top layer, which includes System UI and System app.

- System UI includes apps related to the basic user interface, such as App Bar and Notification. These apps are usually implemented using QML.
- System app includes Settings app and web browser. The settings app is used to control the system properties and is implemented using Enact. As a web browser, an enact browser is used.

7.3.2 Application Framework

To help developers create better applications and services, webOS OSE offers more advanced options and environments as compared to developing solely with HTML5, JavaScript, or CSS. It provides the Enact web app framework and Software Development Kit (SDK).

- **Enact** is a web app framework designed for developing web apps for webOS OSE. For more information and structure of Enact, refer to the Enact developer site.
- The webOS OSE **SDK** offers a development environment and tools including Command-Line Interface, Emulator, Beanviser, and Workflow Designer for web apps and services.

7.3.3 Managers & Services

The managers and services layer includes the following key components.

- **System and Application Manager** (SAM) oversees the behavior of apps. SAM manages each app throughout its lifecycle, including the installation, launch, termination, and removal of the app.
- **Web Application Manager** (WAM) is responsible for launching and managing web apps. WAM optimizes CPU usage, monitors status, and manages access privileges based on web app status.
- **The Activity Manager** is a tool designed to manage and execute activities that are requested from services. An activity is requested with a specific condition, and it gets triggered when that condition is met. You can customize the activity to perform various tasks such as automatic execution of specific services, callback requests, and more.
- **The Luna Surface Manager** (LSM) is a crucial component that serves as a graphics and window manager. LSM is responsible for displaying graphical elements on the screen, managing the composition of these elements, and handling input events for devices like the keyboard and pointer. Additionally, LSM is in charge of managing the System UI, including the App Bar and Notification. To implement the LSM, the Qt framework is used, while QML is the language used for implementing the System UI.
- **uMediaServer** (uMS) is a module that serves as a webOS media framework server. uMS provides interfaces for media playback, manages resources and pipelines, manages policies, and controls the lifecycle of the media player.
- **DB8** is a JSON database that can be embedded and utilized to store and retrieve data in a key-value format. In webOS OSE, LevelDB acts as the backend database for DB8. By default, DB8 is associated with the com.webos.service.db service. Additionally, the com.webos.service.tempdb service is also available, which allows users to configure temporary storage in memory.

7.3.4 Base Components

The base Components layer includes the following key components.

- **LS2**, also called Luna Bus, is a system bus used by webOS OSE as the IPC mechanism used between components in webOS. It is composed of **a client library & Central hub daemon**
- webOS OSE uses Chromium as its web engine. A **web engine** is responsible for loading and parsing web applications or pages made up of HTML, CSS, and JavaScript. It also performs all the required tasks to display the web app on the screen, including layering and rendering. Currently, the web runtime environment of WebOS OSE follows the multi-process model of Chromium, which means that each web app runs as a separate, independent process.
- To help you develop services using JavaScript language, we provide a service framework based on Node.js. **Node.js** is a JavaScript framework that is usually run on a server. We have integrated it into webOS OSE to make it easier for you to develop services. To learn more about developing services using Node.js, please refer to the JS Services documentation.

7.3.5 BSP/Kernel

The BSP/Kernel layer consists of several components to support Raspberry Pi 4. Typically, it provides evdev for event processing, Mesa for graphics support, and 100Base-T, wpa_supplicant, and Bluetooth for connectivity.

CHAPTER 8

QT/QML

QT is a cross-platform application development framework that allows developers to create software applications for desktop, mobile, and embedded systems. It was originally developed by Nokia in 1991 and has since been acquired by The QT Company.

QT provides developers with a wide range of tools, libraries, and modules that can be used to create complex and high-performance applications. It includes a rich set of user interface controls, support for a variety of programming languages, and an extensive set of APIs for data access, networking, multimedia, and more.

One of the main benefits of using QT for application development is its cross-platform capability. Developers can write their code once and deploy it on multiple platforms, including Windows, macOS, Linux, iOS, and Android. This reduces development time and cost and also allows for a wider distribution of the application.

QT also offers a flexible licensing model. Developers can choose from open-source or commercial licenses depending on their needs. This allows for a more customized approach to application development, enabling developers to choose the licensing that is best suited for their applications.

Overall, QT is a powerful and versatile framework that has been used to create a wide range of software applications, from small desktop utilities to large-scale enterprise systems. Its cross-platform support and flexible licensing make it a popular choice among developers across different industries.

The QT framework contains a comprehensive set of highly intuitive and modularized C++ library classes and is loaded with APIs to simplify your application development. QT produces highly readable, easily maintainable, and reusable code with high runtime performance and a small footprint – and it's cross-platform. QT is full of tools to simplify developer’s lives and help with not just coding but also tasks like building, compiling, testing, localization, internationalization, and more.

8.1 Brief of QT Modules

- **QT Core:** Core non-graphical classes used by other modules.
- **QT D-Bus:** Classes for inter-process communication over the D-Bus protocol.
- **QT GUI:** Base classes for graphical user interfaces.
- **QT Network:** Classes to make network programming easier and more portable
- **QT QML:** Classes for QML and JavaScript languages.
- **QT Quick:** A declarative framework for building dynamic applications with custom user interfaces
- **QT Quick Controls:** Lightweight QML types for creating performant user interfaces for desktop, embedded, and mobile devices.
- **QT Quick Dialogs:** Types for creating and interacting with system dialogs in a QT Quick application.
- **QT Quick Layouts:** Layouts for arranging QT Quick items in the user interface.
- **QT Quick Test:** A unit test framework for QML applications, where test cases are written as JavaScript functions.
- **QT Test:** Classes for unit testing QT applications and libraries.
- **QT Widgets:** Classes to extend QT GUI with C++ widgets.
- **Active QT:** Classes for applications that use ActiveX and Component Object Model (COM)
- **QT 3D:** Functionality for near-Realtime simulation systems with support for 2D and 3D rendering.
- **QT 5 Core Compatibility APIs:** QT Core APIs that were in QT 5 but not QT 6.
- **QT Bluetooth:** Provides access to Bluetooth hardware.
- **QT Charts:** UI Components for displaying visually pleasing charts, driven by static or dynamic data models.
- **QT Concurrent:** Classes for writing multi-threaded programs without using low-level threading primitives
- **QT Data Visualization:** UI Components for creating stunning 3D data visualizations.
- **QT Help:** Classes for integrating documentation into applications.
- **QT Image Formats:** Plugins for additional image formats: TIFF, MNG, TGA, WBMP.

- **QT Lottie Animation:** A QML API for rendering graphics and animations in JSON format, exported by the plugin for Adobe After Effects.
- **QT OpenGL:** C++ classes that make it easy to use OpenGL in QT applications.
- **QT Multimedia:** A rich set of QML types and C++ classes to handle multimedia content. Also includes APIs for accessing camera hardware.
- **QT Network Authorization:** Provides support for OAuth-based authorization to online services
- **QT NFC:** Provides access to Near-Field communication (NFC) hardware.
- **QT Positioning:** Provides access to position, satellite info and area monitoring classes.
- **QT Print Support:** Classes to make printing easier and more portable.
- **QT Quick 3D:** Provides a high-level API for creating 3D content or UIs based on QT Quick
- **QT Quick Timeline:** Enables keyframe-based animations and parameterization.
- **QT Quick Widgets:** Provides a C++ widget class for displaying a QT Quick user interface.
- **QT Remote Objects:** Provides an easy-to-use mechanism for sharing a QObject's API (Properties/Signals/Slots) between processes or devices
- **QT SCXML:** Provides classes and tools for creating state machines from SCXML files and embedding them in applications.
- **QT Sensors:** Provides access to sensor hardware.
- **QT Serial Bus:** Provides access to serial industrial bus interfaces. Currently, the module supports the CAN bus and Modbus protocols.
- **QT Serial Port:** Provides classes to interact with hardware and virtual serial ports.
- **QT Shader Tools:** Tools for the cross-platform QT shader pipeline, enabling the use of graphics and compute shaders in QT Quick and other components in the QT ecosystem.
- **QT SQL:** Classes for database integration using SQL.
- **QT State Machine:** Provides classes for creating and executing state graphs.
- **QT SVG:** Classes for displaying the contents of SVG files. Supports a subset of the SVG 1.2 Tiny standard.
- **QT UI Tools:** Classes for loading QWidget based forms created in QT Designer dynamically, at runtime.

- **QT Wayland Compositor:** Provides a framework to develop a Wayland compositor.
- **QT WebChannel:** Provides access to QObject or QML objects from HTML clients for seamless integration of QT applications with HTML/JavaScript clients.
- **QT WebEngine:** Classes and functions for embedding web content in applications using the Chromium browser project.
- **QT WebSockets:** Provides WebSocket communication complaint with RFC 6455.
- **QT WebView:** Displays web content in a QML application by using APIs native to the platform, without the need to include a full web browser stack.
- **QT Virtual Keyboard:** A framework for implementing different input methods as well as a QML virtual keyboard.
- **QT XML:** Provides implementations of the SAX and DOM standards for XML.
- **QT Quick TreeView:** Provides a control that can be used to show a tree model in QT Quick
- **QT Quick Calendar:** Provides a set of types that can be used to build calendars in QT Quick
- **QT Quick MultiEffect:** Provides a QT Quick component for fast, animated effects.
- **QT Digital Advertising:** Provides a lightweight API to incorporate advertisements into QT Quick applications.
- **QT VNC Server:** Provides an API for creating a simple VNC-compatible server with QT.

8.2 Qt Core

The QT Core module adds these features to C++:

- A very powerful mechanism for seamless object communication called signals and slots
- Queryable and designable object properties
- Hierarchical and Queryable object trees that naturally organize object ownership with guarded pointers (QPointer)
- A dynamic cast that works across library boundaries

8.2.1 The Meta-Object System

QT's meta-object system provides the signals and slots mechanism for inter-object communication, run-time type information, and the dynamic property system.

The meta-object system is based on three things:

- The QObject class provides a base class for objects that can take advantage of the metaobject system.
- The Q_OBJECT macro inside the private section of the class declaration is used to enable meta-object features, such as dynamic properties, signals, and slots.
- The Meta-Object Compiler (Moc) supplies each QObject subclass with the necessary code to implement meta-object features.

The Moc tool reads a C++ source file. If it finds one or more class declarations that contain the Q_OBJECT macro, it produces another C++ source file that contains the meta-object code for each of those classes. This generated source is either #include'd into the class's source file or, more usually, compiled and linked with the class's implementation.

In addition to providing the signals and slots mechanism for communication between objects (the main reason for introducing the system), the meta-object code provides the following additional features:

- QObject::metaObject() returns the associated meta-object for the class.
- QMetaObject::className() returns the class name as a string at run-time, without requiring native run-time type information (RTTI) support through the C++ compiler.
- QObject::inherits () function returns whether an object is an instance of a class that inherits a specified class within the QObject inheritance tree.
- QObject::tr () translates strings for internationalization.
- QObject::setProperty() and QObject::property () dynamically set and get properties by name.
- QMetaObject::newInstance() constructs a new instance of the class.

8.2.2 The Property System

QT provides a sophisticated property system similar to the ones supplied by some compiler vendors. However, as a compiler- and platform-independent library, QT does not rely on nonstandard compiler features like __property or [property]. The QT solution works with any standard C++ compiler on every platform QT supports. It is based on the Meta-Object System that also provides inter-object communication via signals and slots.

8.2.3 Object Model

The standard C++ object model provides very efficient runtime support for the object paradigm. But its static nature is inflexible in certain problem domains. Graphical user interface programming is a domain that requires both runtime efficiency and a high level of flexibility. QT provides this by combining the speed of C++ with the flexibility of the QT Object Model.

QT adds these features to C++:

- A very powerful mechanism for seamless object communication called signals and slots
- Queryable and designable object properties
- Powerful events and event filters
- Contextual string translation for internationalization
- Sophisticated interval driven timers that make it possible to elegantly integrate many tasks in an event-driven GUI
- Hierarchical and queryable object trees that organize object ownership in a natural way
- Guarded pointers (QPointer) that are automatically set to 0 when the referenced object is destroyed, unlike normal C++ pointers which become dangling pointers when their objects are destroyed
- A dynamic cast that works across library boundaries.
- Support for custom type creation.

8.2.4 Object Trees & Ownership

QObjects organize themselves in object trees. When you create a QObject with another object as parent, it's added to the parent's children () list, and is deleted when the parent is. It turns out that this approach fits the needs of GUI objects very well. For example, a QShortcut (keyboard shortcut) is a child of the relevant window, so when the user closes that window, the shortcut is deleted too. QQuickItem, the basic visual element of the QT Quick module, is inherited from QObject, but has a concept of the visual parent which differs from that of the QObject parent. An item's visual parent may not necessarily be the same as its object parent. See Concepts - Visual Parent in QT Quick for more details. QWidget, the fundamental class of the QT Widgets module, extends the parent-child relationship. A child normally also becomes a child widget, i.e., it is displayed in its parent's coordinate system and is graphically clipped by its parent's boundaries. For example, when the application deletes a message box after it has been closed, the message box's buttons and label are also deleted, just as we'd want, because the buttons and label are children of the message box.

You can also delete child objects yourself, and they will remove themselves from their parents. For example, when the user removes a toolbar, it may lead to the application deleting one of its QToolBar objects, in which case the tool bar's QMainWindow parent would detect the change and reconfigure its screen space accordingly.

The debugging functions QObject::dumpObjectTree() and QObject::dumpObjectInfo() are often useful when an application looks or acts strangely.

8.3 Qt Quick

The QT Quick module is the standard library for writing QML applications. While the QT QML module provides the QML engine and language infrastructure, the QT Quick module provides all the basic types necessary for creating user interfaces with QML. It provides a visual canvas and includes types for creating and animating visual components, receiving user input, creating data models and views and delayed object instantiation.
The QT Quick module provides both a QML API, which supplies QML types for creating user interfaces with the QML language, and a C++ API for extending QML applications with C++ code.

CHAPTER 9
Qt Quick Scene Graph

QT Quick is a framework for creating dynamic, fluid user interfaces in QT applications. It provides a declarative language (QML) for designing UIs, and a powerful engine (the QT Quick Scene Graph) for rendering them. The Scene Graph is the underlying technology that makes it possible to create efficient, hardware-accelerated UIs with complex animations and smooth transitions.

At its core, the Scene Graph is a graph-based representation of a QT Quick UI. It is designed to be highly optimized for performance and to take advantage of modern graphics hardware to provide smooth, responsive UIs. When you create a QT Quick application, the engine automatically builds a Scene Graph for you based on the QML code you provide.

The Scene Graph is built up of a hierarchy of nodes, each of which represents a different part of the UI. For example, a node might represent a rectangle, an image, or a text label. Nodes can be nested inside other nodes, forming a tree-like structure that represents the entire UI.

Each node in the Scene Graph has its own set of properties, which can be set either in QML or in C++. These properties control things like the position, size, color, and opacity of the node. They can also be used to apply transformations (such as scaling or rotating) to the node or to define custom animations and transitions.

The Scene Graph is designed to be highly modular and extensible so that it can be customized to fit the needs of a wide range of applications. For example, you can create custom nodes that implement specific UI elements, or you can create custom shaders to apply special effects to your UI.

Overall, the QT Quick Scene Graph is a powerful and flexible technology that makes it possible to create highly optimized, hardware-accelerated UIs with complex animations and transitions. Whether you are building a desktop application, a mobile app, or a game, Scene Graph can help you create a UI that is both beautiful and responsive.

8.1 The Scene Graph in Qt

QT Quick 2 makes use of a dedicated scene graph that is then traversed and rendered via a graphics API such as OpenGL ES, OpenGL, Vulkan, Metal, or Direct 3D. Using a scene graph for graphics rather than the traditional imperative painting systems (QPainter and similar) means the scene to be rendered can be retained between frames, and the complete set of primitives to render is known before rendering starts. This opens up several optimizations, such as batch rendering to minimize state changes and discarding obscured primitives.

For example, a user interface contains a list of ten items where each item has a background color, an icon, and a text. Using the traditional drawing techniques would result in 30 draw calls and a similar amount of state changes. A scene graph, on the other hand, could reorganize the primitives to render such that all backgrounds are drawn in one call, then all icons, then all the text, reducing the total amount of draw calls to only 3. Batching and state change reduction like this can greatly improve performance on some hardware.

The scene graph is closely tied to QT Quick 2.0 and cannot be used stand-alone. The scene graph is managed and rendered by the QQuickWindow class and custom Item types can add their graphical primitives into the scene graph through a call to QQuickItem::updatePaintNode().

The scene graph is a graphical representation of the Item scene, an independent structure that contains enough information to render all the items. Once it has been set up, it can be manipulated and rendered independently of the state of the items. On many platforms, the scene graph will even be rendered on a dedicated render thread while the GUI thread is preparing the next frame's state.

Note: Much of the information listed on this page is specific to the built-in, default behavior of the QT Quick Scene graph. When using an alternative scene graph adaptation, such as, the software adaptation, not all concepts may apply. For more information about the different scene graph adaptations see Scene Graph Adaptations.

The QT Quick Scene Graph is a powerful rendering engine that is used to render the user interfaces of QT Quick applications. It is designed to be highly optimized for performance and to take advantage of modern graphics hardware to provide smooth, responsive UIs. When you create a QT Quick application, the engine automatically builds a Scene Graph for you based on the QML code you provide.

The Scene Graph is built up of a hierarchy of nodes, each of which represents a different part of the UI. For example, a node might represent a rectangle, an image, or a text label. Nodes can be nested inside other nodes, forming a tree-like structure that represents the entire UI.

Each node in the Scene Graph has its own set of properties, which can be set either in QML or in C++. These properties control things like the position, size, color, and opacity of the node. They can also be used to apply transformations (such as scaling or rotating) to the node or to define custom animations and transitions. The Scene Graph is designed to be highly modular and extensible so that it can be customized to fit the needs of a wide range of applications. For example, you can create custom nodes that implement specific UI elements, or you can create custom shaders to apply special effects to your UI.

Node Types: The Scene Graph supports a wide range of node types, each of which represents a different type of UI element. Some of the most commonly used node types include:

- Rectangle: A rectangular area with a solid color or a border.
- Image: An image or icon.
- Text: A text label.
- Item: A container for other nodes.
- Transform A node that applies a transformation (such as scaling or rotating) to its child nodes.

Nodes can be combined in various ways to create complex UIs. For example, you might use a Rectangle node as the background of a UI, and then add Text and Image nodes on top of it to create a more complex layout.

Properties

Each node in the Scene Graph has a set of properties that control its appearance and behavior. Some of the most commonly used properties include:

- X and Y: The position of the node relative to its parent.
- Width and Height: The size of the node.
- Color: The color of the node.
- Opacity: The opacity of the node (how transparent it is).
- Transform A transformation (such as scaling or rotating) that is applied to the node.

Transformations: Transformations are a powerful feature of the Scene Graph that allow you to apply various transformations (such as scaling or rotating) to a node or group of nodes. Transformations are represented by the Transform node type. Here's an example of how you might use a Transform node to rotate an Image node:

```
Image
{
    id: image
    source: "image.png"
    Transform
   {
       rotation: 45
   }
}
```

In this example, we've added a Transform node as a child of the Image node and set its rotation property to 45 degrees. This will cause the image to be rotated by 45 degrees when it is rendered.

8.1.1 QSGNodeQQuickItem::updatePaintNode( )

The QSGNodeQQuickItem::updatePaintNode() function is a key method in the QT Quick Scene Graph, used to update the visual representation of a QQuickItem when its properties change. In this function, you can create or modify a QSGNode that represents the visual appearance of the QQuickItem, and return it to the Scene Graph for rendering.

The updatePaintNode() function is called whenever a QQuickItem needs to be redrawn, which can happen for various reasons. For example, if the size or position of the item changes, or if its content changes, the Scene Graph will call this function to update its appearance.

The function takes two arguments: an old QSGNode (representing the previous visual appearance of the item), and an UpdatePaintNodeData structure containing information about the current state of the item (such as its position and size). The function should return a new QSGNode that represents the updated appearance of the item.

The first step in the updatePaintNode() function is to check whether the old node can be reused, or whether a new node needs to be created. If the old node is of the correct type and can be updated to reflect the new state of the item, you can modify it and return it as the new node. Otherwise, you need to create a new node from scratch.

Once you have a node, you can set its properties to reflect the current state of the item. For example, you might set the node's position, size, color, or opacity properties based on the item's properties. You can also add child nodes to the node to represent any sub-elements of the item (such as text labels or images).

One important consideration when updating a QSGNode is to minimize the amount of work required to render it. The Scene Graph is designed to be highly optimized for performance, and 25 to take advantage of modern graphics hardware to provide smooth, responsive UIs. To achieve this, it uses various techniques such as batching and caching to minimize the number of rendering operations that need to be performed.

To take advantage of these optimizations, you should try to keep the structure of your QSGNodes as simple and efficient as possible. This means avoiding unnecessary nodes or transformations and grouping similar elements together to reduce the number of draw calls.

In summary, the QSGNodeQQuickItem::updatePaintNode() function is a critical part of the QT Quick Scene Graph, used to update the visual appearance of QQuickItems in response to changes in their properties. By creating or modifying QSGNodes that represent the visual appearance of the items, you can ensure that the Scene Graph renders your UI efficiently and accurately, providing a smooth and responsive user experience.

Called on the render thread when it is time to sync the state of the item with the scene graph.

The function is called as a result of QQuickItem::update (), if the user has set the QQuickItem::ItemHasContents flag on the item.

The function should return the root of the scene graph subtree for this item. Most implementations will return a single QSGGeometryNode containing the visual representation of this item. oldNode is the node that was returned the last time the function was called. updatePaintNodeData provides a pointer to the QSGTransformNode associated with this QQuickItem.

```
QSGNode *MyItem::updatePaintNode(QSGNode *node, UpdatePaintNodeData *)
{
    QSGSimpleRectNode *n = static_cast(node);
    if (!n)
   {
        n = new QSGSimpleRectNode();
        n->setColor(QT::red);
    }
    n->setRect(boundingRect());
    return n;
}
```

The main thread is blocked while this function is executed so it is safe to read values from the QQuickItem instance and other objects in the main thread.

If no call to QQuickItem::updatePaintNode() results in actual scene graph changes, like QSGNode::markDirty() or adding and removing nodes, then the underlying implementation may decide to not render the scene again as the visual outcome is identical.

Warning: It is crucial that graphics operations and interaction with the scene graph happens exclusively on the render thread, primarily during the QQuickItem::updatePaintNode() call. The best rule of thumb is to only use classes with the "QSG" prefix inside the QQuickItem::updatePaintNode() function.

Warning: This function is called on the render thread. This means any QObjects or thread local storage that is created will have affinity to the render thread, so apply caution when doing anything other than rendering in this function. Similarly for signals, these will be emitted on the render thread and will thus often be delivered via queued connections.

Note: All classes with QSG prefix should be used solely on the scene graph's rendering thread. See Scene Graph and Rendering for more information.

CHAPTER 10
CONCLUSION

In Conclusion, there is a growing trend of cloud gaming and an increasing number of users engaging in this form of entertainment. In response to this trend, the proposed integration of gamepad support for webOS TV is a strategic move. Although webOS TV has been updated with various features to accommodate diverse input devices, it still lacks the ability for users to control the TV using gamepad controllers. The current scenario, where users find themselves navigating both the TV Remote Control Unit (RCU) and a gamepad controller while playing cloud games on webOS TV, poses a significant usability challenge. This limitation impacts the overall user experience and introduces complexities that can be cumbersome for users. The introduction of gamepad support addresses this challenge head-on. By enabling users to seamlessly control both webOS TV and cloud gaming with gamepad controllers, the proposed solution aims to streamline the user experience. This enhancement not only alleviates the current navigation challenges but also aligns with the evolving preferences of users engaging in cloud gaming.

REFERENCES

1. https://www.youtube.com/playlist?list=PL6CJYn40gN6hdNC1IGQZfVI707dh9DPRc [QML Language for Frontend]

2. https://doc.qt.io/qt-6/qtqml-index.html [QT Language for Backend]

3. https://github.com/spotify/linux/blob/master/include/linux/input.h [Key Mapping Information]

4. https://www.webosose.org/docs/home/ [webOS Documentation]

5. https://doc.qt.io/qt-6/qtqml-index.html [QT/QML Documentation]

6. https://doc.qt.io/qt-6/qsgmaterial.html [Qt Scene Graph Documentation]