White Paper for Tizen Platform Developers and Manufacturers

Table of Contents

Overview

This document is a guide for Tizen platform developers and manufacturers building Tizen-based devices. The content is based on the Tizen 2.3 alpha version.

This document provides information on how to boot Tizen on new hardware and to create products based on the Tizen OS. It covers the porting process in detail by elaborating the Tizen architecture, needed tools, and the development environment set-up, as well as demonstrating how you can create a Tizen Image and perform the modifications needed across various functional areas.

1. Introduction to Tizen

Tizen is a standards-based platform that provides Web and native APIs for developing applications for multiple device categories. Tizen is currently targeted for mobile, wearable, TV, IVI, and camera devices, and an expansion to further device categories is planned for the future.

1.1 Tizen Architecture

The following figure illustrates the Tizen architecture.

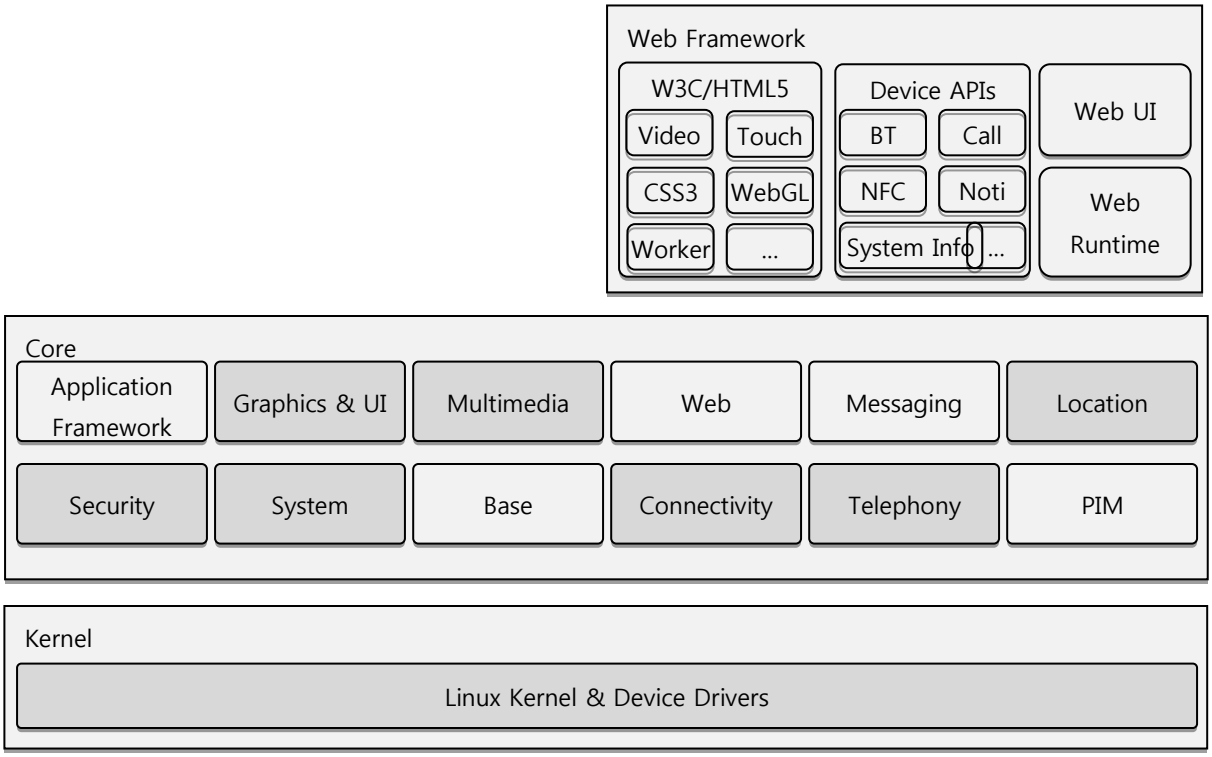

Figure 1. Tizen architecture

Application Framework

The Application Framework provides application management, including launching other applications using the package name, URI, or MIME type. It also launches predefined services, such as the system dialer application. The Application Framework notifies applications of common events, such as low memory, low battery, screen orientation changes, and push notifications.

Base

The Base module contains Linux base essential system libraries that provide key features, such as database support, internationalization, and XML parsing.

Connectivity

The Connectivity module consists of all network- and connectivity-related functionalities, such as 3G, Wi-Fi, Bluetooth, HTTP, and NFC (Near Field Communication). Data network is based on ConnMan (Connection manager), which provides 3G- and Wi-Fi-based network connection management.

Graphics and UI

For the mobile profile, the Graphics and UI module consist of the system graphic and UI stacks, which are called the “Native Framework”. The module internally utilizes EFL (Enlightenment Foundation Libraries) for X11-based window management and phone indicator. Specialized control for fluent animations is also available through EFL's elementary framework. The Native Framework has various built-in input methods and OpenGL® ES APIs.

The module provides WebKit-based graphics capable of running within a full browser UI or a dedicated Web Runtime (without browser window), all based on Tizen's own HTML5 canvas WebKitEFL implementation. Support further covers WebGL and Web-based frameworks for UI, such as jQuery Mobile (helpful in porting existing jQuery code).

Location

The Location module provides location-based services (LBS), including position information, geocoding, satellite information, and GPS status. It delivers location information from various positioning sources, such as GPS, WPS (Wi-Fi Positioning System), Cell ID, and sensors.

Messaging

The Messaging module provides SMS, MMS, email, and IM features.

Multimedia

The Multimedia Framework provides various multimedia functionality. It consists of the Player/Streaming Framework, Camera/Recording Framework, Media Content Framework, and Screen Mirroring Framework.

PIM (Personal Information Management)

The PIM module enables user data management on the device. It allows you to manage calendar, contacts, and tasks, and retrieve data about the device context (such as device position and cable status).

Security

The Security module is responsible for security deployment across the system. It consists of platform security enablers, such as access control, certificate management, and Anti-virus framework.

System

The System module consists of system and device management features, including:

- Interfaces for accessing devices, such as sensors, display, or vibrator.
- Power management, such as LCD display backlight changes and application processor sleep.
- Device monitoring and event handling in case of, for example, USB, MMC, charger, and ear jack events.

- System upgrade.
- Mobile device management.

Telephony

The Telephony module consists of cellular functionalities communicating with the modem:

- Managing call-related and non-call-related information and services for UMTS and CDMA.
- Managing packet service and network status information for UMTS and CDMA.
- Managing SMS-related services for UMTS and CDMA.
- Managing SIM files, phone book, and security.
- Managing SIM Application Toolkit services for UMTS.

Web

The Web module provides a complete implementation of the Tizen Web API optimized for low-power devices. It includes WebKit, which is a layout engine designed to allow Web browsers to render Web pages. It also provides Web runtimes for Web applications.

2. Getting the Source Code and Making Builds

This chapter describes how you can retrieve the source code and make the platform, kernel, and image builds.

2.1 Getting the Source Code

To get the source code:

1. Confirm the package name from the Tizen Gerrit Project List (https://review.tizen.org/gerrit/#/admin/projects). You can also use the following command:
   ```
   $ ssh review.tizen.org gerrit ls-projects
   ```

2. Clone the Gerrit Project with the following command:
   ```
   $ git clone [-b <Branch>] ssh://<Username>@review.tizen.org:29418/<Gerrit_Project> [<Local_Project>]
   ```

   For example:
   ```
   $ git clone ssh://<Username>@review.tizen.org:29418/platform/upstream/dbus
   ```

To clone all projects, read the instructions in https://source.tizen.org/documentation/developer-guide/getting-started-guide/cloning-tizen-source.

2.2 Making Platform Builds

To make the platform build:

1. To get access, register for an account at https://www.tizen.org/user/register.

2. Follow the Tizen platform development workflow as defined in the following figure.

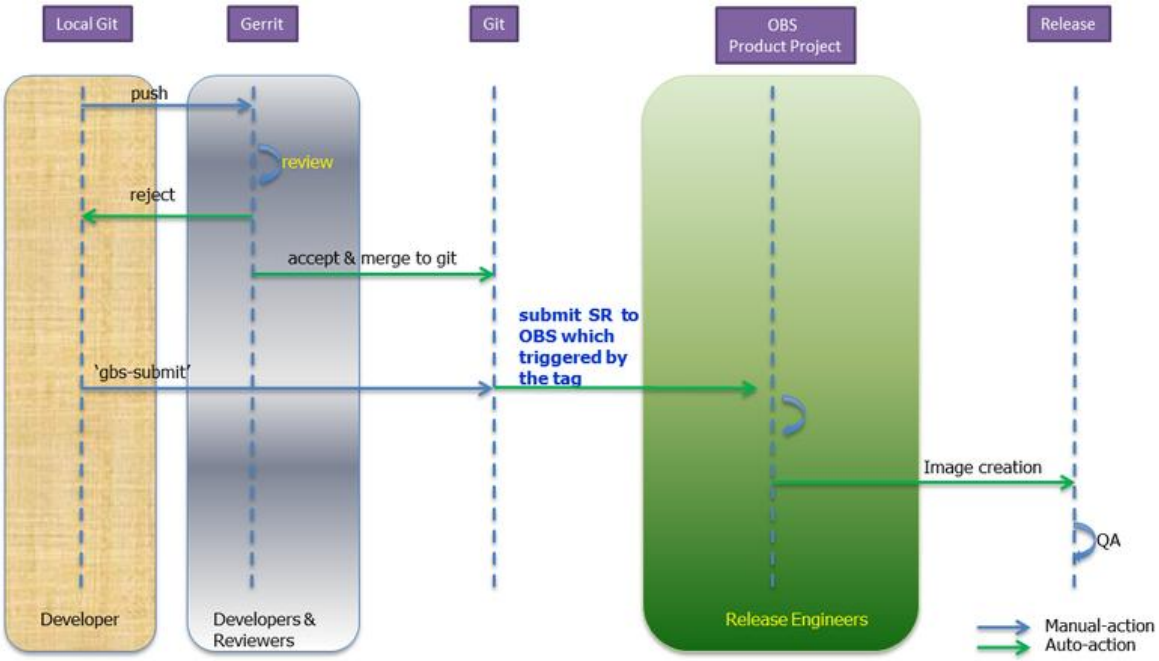

Figure 2. Tizen platform development workflow

Git

Git is a particularly powerful, flexible, and low-overhead version control system that makes collaborative development efficient and robust. For more information, see:

- Git Community Book: http://git-scm.com/documentation
- Git Wiki: https://git.wiki.kernel.org/index.php/Main_Page
- Git Manual Page: https://git.wiki.kernel.org/index.php/Main_Page

Gerrit

Gerrit is a Web-based code review system, facilitating online code reviews for projects using the Git version control system. By showing changes in a side-by-side display and supporting inline comments, Gerrit optimizes the code review process, enhancing review quality. Furthermore, by permitting any authorized user to submit changes to the central Git repository, Gerrit simplifies the maintenance of Git-based projects, enabling a more centralized use of Git. For more information, see:

Gerrit Document Page: https://review.tizen.org/gerrit/Documentation/index.html

Submit Request (SR)

After changes are submitted to the central Git repository, you can submit your changes to the OBS by using “gbs submit” command. The command will create and send a special Git tag and it will trigger builds in OBS.

OBS

Open Build Service (OBS) is an open and complete distribution development platform that provides the infrastructure for developers to easily create and release open source software for various Linux distributions on different hardware architectures. In addition, the OBS delivers a collaborative environment that enables developer groups to build and submit changes to other projects. For more information, see:

- Open Build Service: http://openbuildservice.org/
- OBS Portal: http://en.opensuse.org/openSUSE:Build_Service

GBS

Tizen developers use the git and gbs command line tools for most of their work. Using the Git Build System (GBS), you can build your git repository locally. For more information, see:

- Local Build with GBS: https://source.tizen.org/documentation/developer-guide/getting-started-guide/building-packages-locally-gbs

MIC

MIC Image Creator allows you to create downloadable binary images. You can also create custom binary images in your local with MIC. For more information, see:

- Creating Tizen Images with MIC: https://source.tizen.org/documentation/developer-guide/getting-started-guide/creating-tizen-images-mic

2.3 Making Kernel Builds

To make the Tizen ARM kernel build:

1. If the target and your host are different (such as x86), install and set up cross-compile tools on your system. You can use their Linarotoolchain binaries or the Ubuntu package to set up your environment for the cross tools (for example, export CROSS_COMPILE=....).

2. Set up the `.config` file for RD-PQ:
   ```
   $ make ARCH=arm trats2_defconfig
   ```

3. After reconfiguring according to your needs (use the `make ARCH=arm menuconfig` command) or using the stock configuration (no modifications), build the kernel:
   ```
   $ make ARCH=arm uImage
   ```

4. Build and make the kernel module image.
   Note: If needed, do sudo first to let sudo -n work in the script.
   ```
   $ sudols
   $ scripts/mkmodimg.sh
   ```

5. Send the images to the target through lthor:
   ```
   $ lthor arch/arm/boot/uImageusr/tmp-mod/modules.img
   ```

   Or make your own tar file from the two files:
   ```
   $ tar cf FILENAME_YOU_WANT.tar -C arch/arm/boot uImage -C ../../../usr/tmp-mod
   modules.img
   ```

2.4 Making Image Builds

To make binary images:

1. Install platform packages into a chrooted temporary directory.

2. Run custom scripts.

3. Create a file system with partition information.

Tizen 2.x

To define how to build the binary images for the profile, 2 special gerrit projects are used:

- tools/package-groups (https://review.tizen.org/gerrit/gitweb?p=tools/package-groups.git)
  Defines the package groups. See the `patterns` directory: each `*.yaml` file defines package groups.

- tools/image-configurations (https://review.tizen.org/gerrit/gitweb?p=tools/image-configurations.git)
  Defines how the binary images can be constructed. Multiple configurations can be defined (`configurations.yaml`):
  - Directories with uppercase letters store each binary configuration.
  - “custom/part” holds partitions.
  - “custom/scripts” holds scripts after installing all platform packages.
  Each binary configuration can hold different package groups. Groups are described with “@<name of package group>”.

3. Tizen System Layer

This chapter provides detailed information on the Tizen system layer.

3.1 Kernel

The kernel is the operating system that drives the platform. In this document, “kernel” refers to the open source Linux kernel that is customized for the Tizen platform. The following sections give a brief overview about the Tizen kernel configuration and environment to customize the kernel.

Kernel Configurations

The following table defines the recommended kernel configurations to be used for the Tizen platform.

Table 1. Kernel configurations

Configuration	Description
CONFIG_CGROUPS/CONFIG_CGROUP_MEM_RES_CTRL	Control group
CONFIG_CMA/CONFIG_DMA_CMA	Contiguous memory allocator
CONFIG_DMA_SHARED_BUFFER	Share a buffer among dma devices
CONFIG_DRM	Direct rendering manager graphic infrastructure
CONFIG_VIDEO_V4L2_SUBDEV_API	Video for Linux of the multimedia API
CONFIG_USB_GADGET	USB gadget support
CONFIG_USB_G_SLP	Tizenusb_mode (sdb, ether) composite driver
CONFIG_ANDROID_LOGGER	Dlog log driver
CONFIG_EXT4_FS	Ext4 file system
CONFIG_SECURITY_SMACK	SMACK security access control

Tizen File System

Tizen adopts the ext4 file system as a default root file system. Unlike Android, Tizen does not use ramdisk and initramfs. The VFAT file system can be used for an external SD card.

Tizen Partition Layout

The following figure illustrates an example of the Tizen partition layout. The product vendor can modify the sequence or partition layout for their devices, as needed.

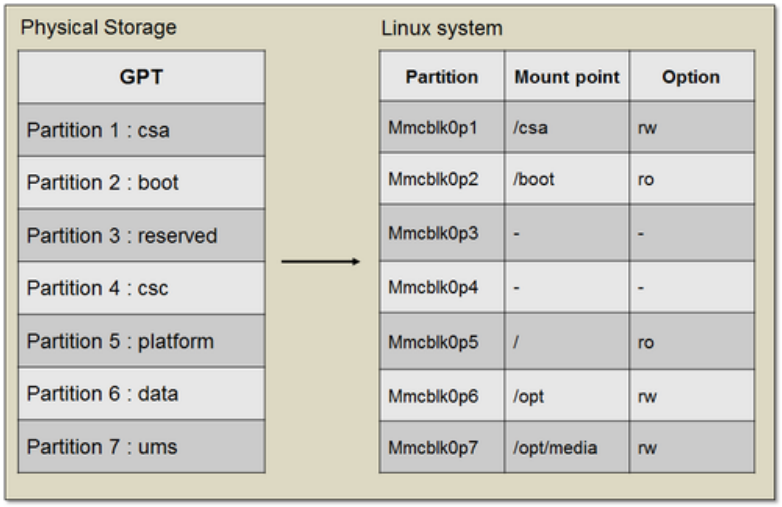

Figure 3. Tizen partition layout

- Partition 1: CSA (Configuration Saved Area)
  Contains non-volatile data that is the calibration value of the modem.

- Partition 2: Boot
  Includes the kernel image modem image and device driver modules.

- Partition 3: Platform
  Contains the root file system, fundamental Tizen frameworks, and some general Linux utility.

- Partition 4: Data
  Includes applications, libraries of applications, and the platform database.

- Partition 5: CSC (Customer Software Configuration)
  Stores the customer's software configuration.

- Partition 6: UMS (USB Mass Storage)
  Contains the default (media) contents.

File System Hierarchy Standard in Tizen

Each partition has the hierarchy illustrated in the following figure.

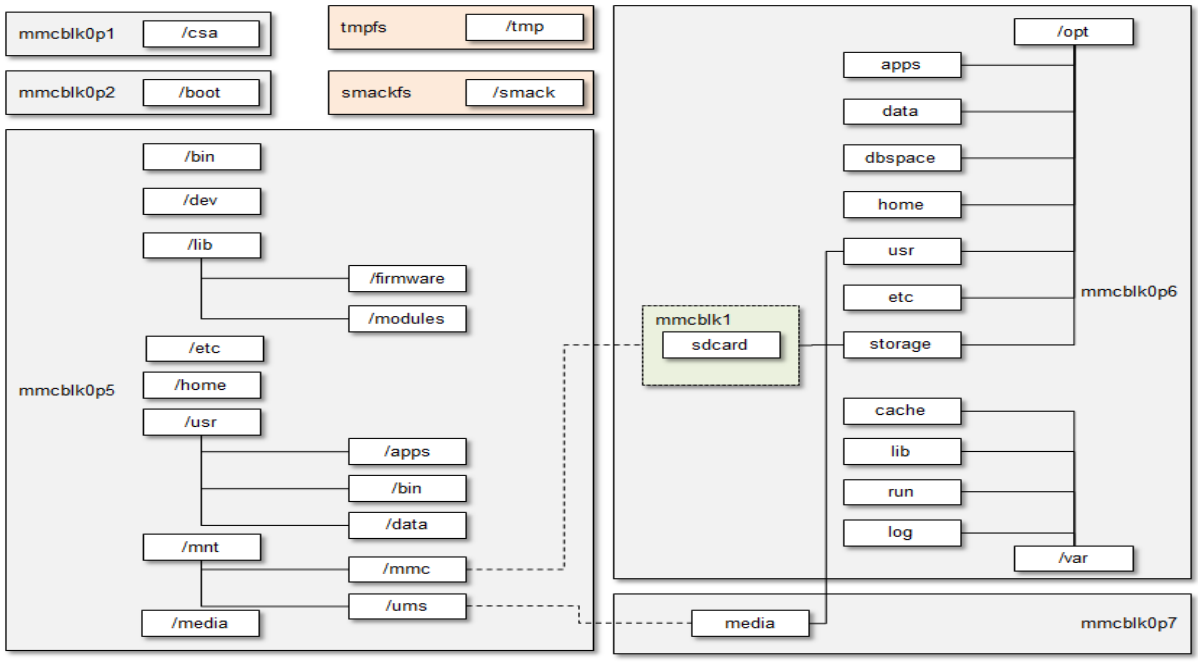

Figure 4. File system hierarchy

- `/usr/apps`: In-house application
- `/opt/usr/apps` (`/opt/apps`): Third-party and downloadable applications
- `/opt/dbspace`: Database files for the Tizen framework
- `/opt/storage/sdcard`: External sdcard mount point
- `/smack`: SMACK file system to load and store smack rules

Memory Management in Tizen

Tizen maintains a process and its resources using cgroup. If a process moves to the foreground from the background, the process group is moved into the “foreground” cgroup.

According to the foreground group policy, the priority of the process is promoted to reduce the possibility that the process is killed in a low memory circumstance. If the system memory reaches the low memory threshold, the kernel notifies the resourced. The resourced reclaims the background group to get sufficient free memory. If the resourced fails to get the free memory, it attempts to kill a victim process which has the highest score (“oom_score_adj” x “memory size of the process”).

3.2 System

This section describes the system and sensor frameworks.

3.2.1 System Framework

The following figure illustrates the System framework.

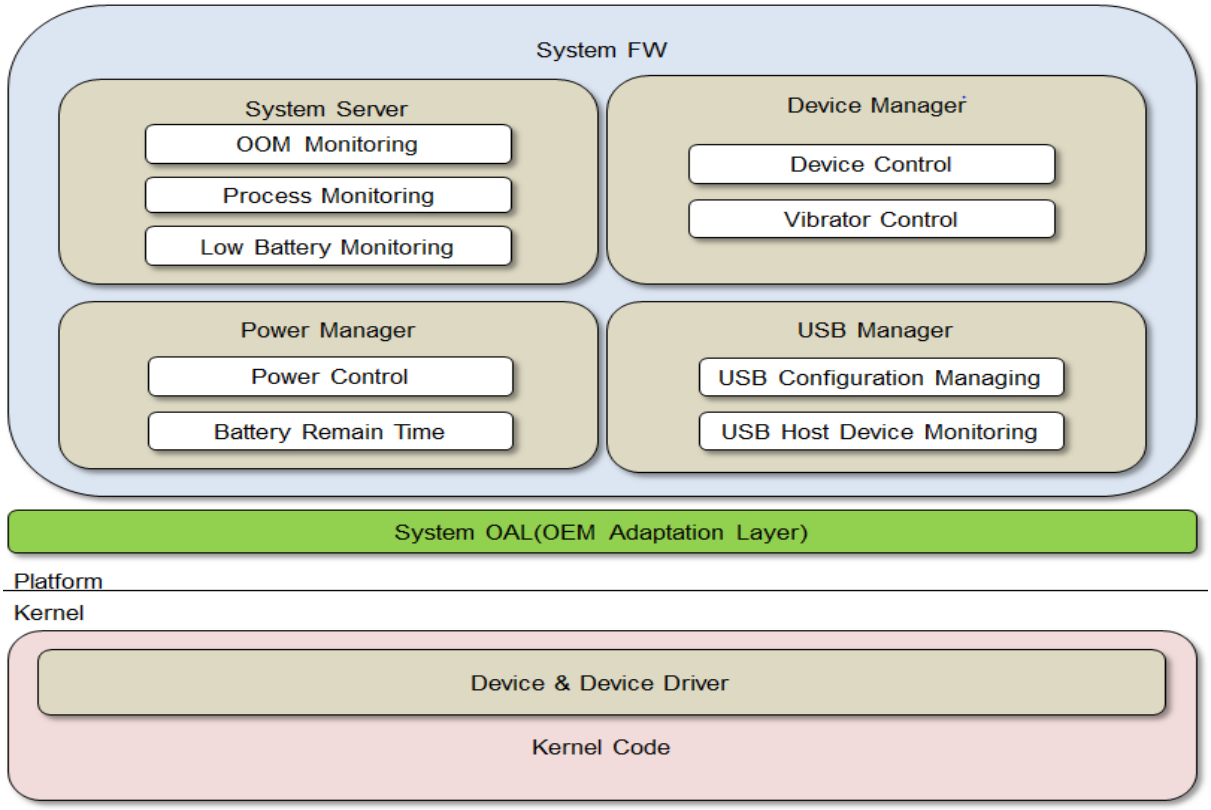

Figure 5. System framework

System server

The system server handles system events, such as out of memory events and changes in the battery level and plug & play device status, as well as managing the process watchdog.

Power manager

The power manager is a session daemon that runs to manage the system power. It provides conditional state transition. The Tizen power manager functionalities control the display backlight changes and device sleep.

Device manager

The device manager provides the interface for controlling all devices, such as LCD backlight changes and application processor sleep.

USB manager

The USB manager sets USB configurations to connect to the PC, and monitors external USB host devices, such as keyboards, mice, cameras, USB storages, and USB printers.

Porting the OAL Interface (Mandatory)

The System framework has the libdeviced library for applications, and the libdevice-node library as a wrapper library of the OAL (OEM Adaption Layer). The OAL interface provides an easy way for devices to access the kernel by grouping OAL APIs depending on the device type. The OAL APIs support all of the devices specified by vendors.

OEM developers must implement the API defined in `devman_plugin_intf.h` and compile their library as `libslp_devman_plugin.so`.

The Devman library uses sysfs for interfaces with device drivers and kernel. sysfs is a virtual file system provided by Linux 2.6 or above.

To configure the OAL interface:

1. Install the OEM library as `libslp_devman_plugin.so` to `/usr/lib`.

2. The OAL API definitions are in the `devman_plugin_intf.h` header file, which is located in libdevice-node.

systemd

systemd is the first process to execute user space. It manages boot sequences and launches deviced, resourced, and other daemons.

The unit files of Tizen are located in `/usr/lib/systemd/system` and `/usr/lib/systemd/user`. You must add your own unit files there.

You must enable 'cgroup' and 'autofs' options in the kernel configuration for systemd. systemd also depends on dbus and some libraries, such as libnotify and libudev.

For more detailed information on system, see the systemd wiki (http://www.freedesktop.org/wiki/Software/systemd/).

3.2.2 Sensor Framework

The Sensor framework provides sensor events to applications and system components. A sensor event is measured information from hardware based on a sensor or a virtual sensor.

Tizen supports individual plugin frameworks for the following sensors:

- Accelerometer sensor
- Gyroscope sensor
- Proximity sensor

- Geomagnetic sensor
- Light sensor

The following figure illustrates the Sensor framework.

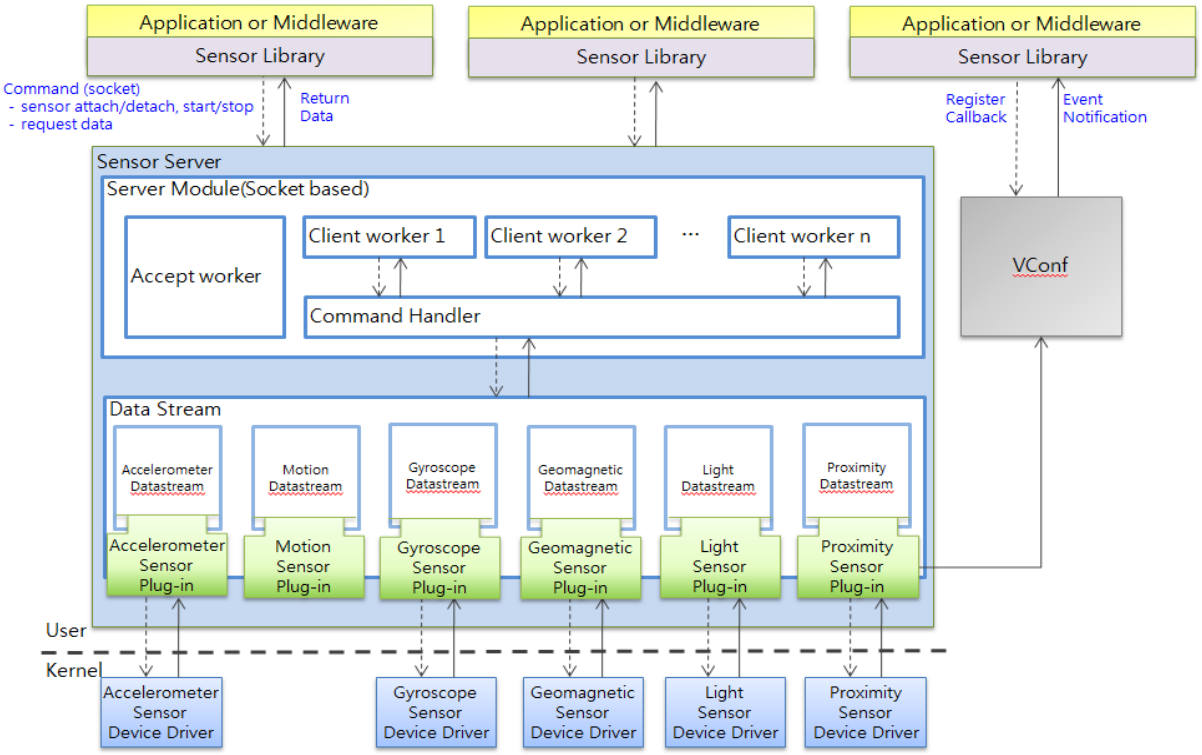

Figure 6. Sensor framework

Porting the OAL Interface (Mandatory)

The sensor OAL includes the processor plugin, the filter plugin, and the sensor plugin. The accelerometer sensor (accel) is used as an example to illustrate each plugin prototype implementation. You can use common sensor plugin (processor, filter, sensor) code in git: `sensor-framework/sensor-plugin-source`.

- Processor plugin: Active component (it has a thread) that processes data or makes events from a filter or from sensor data
- Sensor plugin: A passive component that gets raw data from the kernel node
- Filter plugin: A passive component that converts sensor raw data to other types of data.

Configuration

The Sensor framework loads the `sf_sensor.conf`, `sf_filter,conf`, `sf_processor.conf`, and `sf_datastream.conf` configuration files for loading sensor, filter, and processor plugins. All configuration files are included in the sensor-framework package.

To add the sensor API and enum to the library, add APIs for your sensor and assign your private number for the event/data type.

For more detailed information on the Sensor framework, see the “Sensor Framework” part of the Tizen wiki site (https://wiki.tizen.org/wiki/Porting_Guide#Description_2).

3.3 Graphics and Window System (OpenGL ES, X Server)

The following figure illustrates the Tizen UI and graphics module architecture.

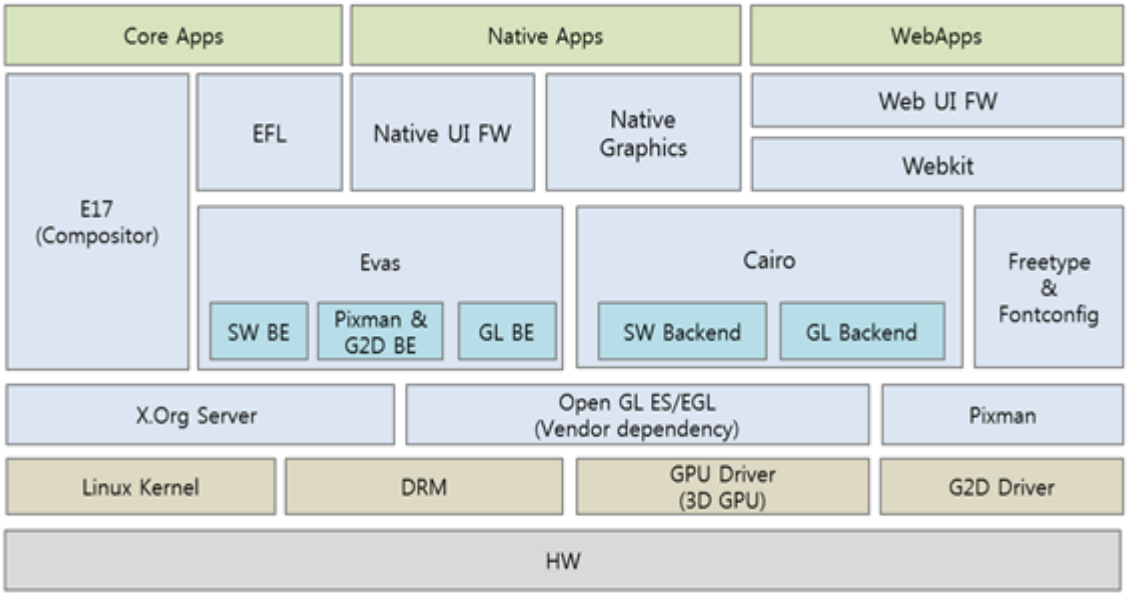

Figure 7. UI and graphics module architecture

Graphics

Tizen provides high-performance 3D graphics as a component of the UI and graphics module. There are hardware-accelerated OpenGL® ES (Open Graphics Library Embedded System) and EGL™ (Embedded-System Graphics Library) features for 3D applications, such as 3D games.

OpenGL® ES is an application programming interface (API) for advanced 3D graphics, targeted at handheld and embedded devices. To overcome device constraints, such as limited processing capabilities and memory availability, it provides a subset of the functionality in OpenGL®.

Embedded system-specific features have been added to enhance rendering efficiency, such as precision qualifiers to the shading language from OpenGL® ES 2.0.

OpenGL ES 3D Graphics Library

The following figure illustrates the interface for Tizen high-performance 3D graphics library with OpenGL® ES.

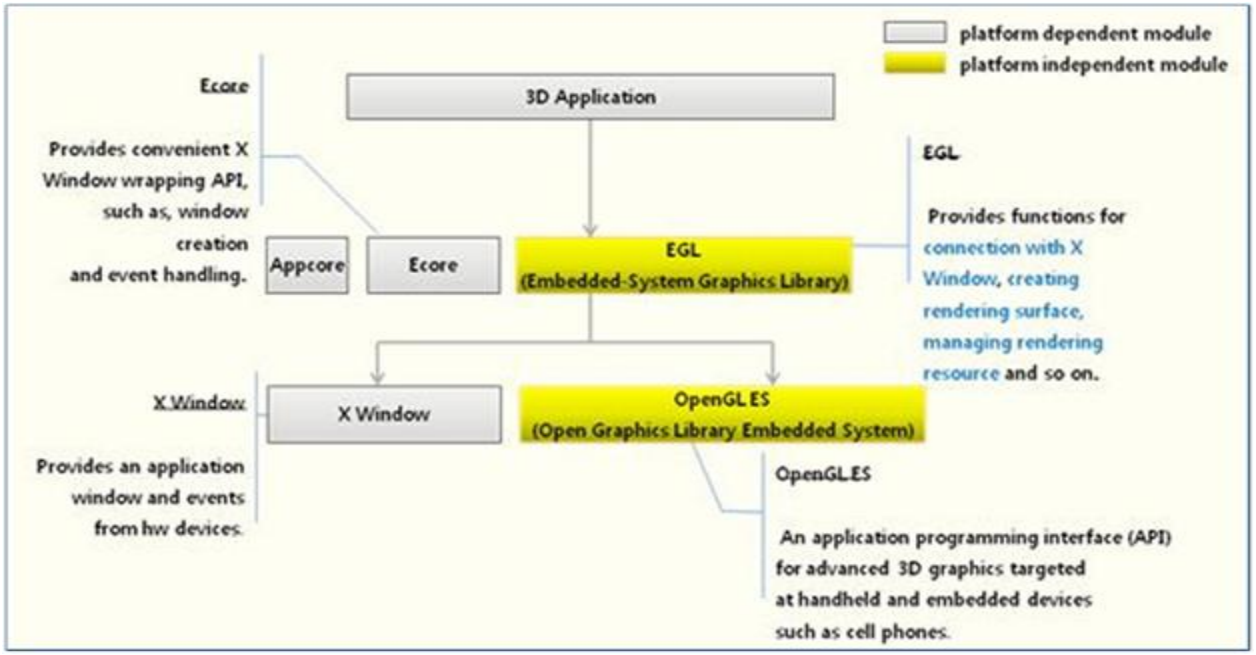

Figure 8. 3D graphics library interface

Porting Requirements

3D graphics vendors must provide the following porting requirements for Tizen to function correctly:

- OpenGL® ES
  - The required version of OpenGL® ES: 1.1 and 2.0
  - The driver must also support the following extensions to OpenGL® ES 1.1:

GL_OES_framebuffer_object	GL_OES_blend_subtract
GL_OES_blend_func_separate	GL_OES_matrix_palette
GL_OES_draw_texture	GL_OES_texture_cube_map
GL_OES_query_matrix	GL_OES_point_size_array

  - The driver must also support the following extensions to OpenGL® ES 2.0:

GL_OES_EGL_image	GL_EXT_texture_format_BGRA8888
GL_OES_get_program_binary	GL_OES_texture_npot
GL_OES_fragment_precision_high	GL_OES_rgb8_rgba8
GL_OES_depth24	GL_OES_vertex_half_float
GL_OES_texture_float	GL_OES_compressed_ETC1_RGB8_texture
GL_OES_packed_depth_stencil	GL_OES_standard_derivatives
GL_OES_element_index_uint	GL_OES_mapbuffer

GL_EXT_multi_draw_arrays	GL_OES_vertex_array_object
GL_IMG_texture_compression_pvrtc	GL_OES_read_format
GL_EXT_multisampled_render_to_texture	

  - Vendors need to provide tools for generating a binary shader on Linux.

- EGL™
  - The minimum version required for EGL™ is 1.4. EGL™ must support DRI2.
  - Common 3D applications use DRI2-based EGL™.
  - The 3D composite window manager requires DRI2-based EGL™.
  - The driver must support the following extensions to EGL™ 1.4:

GL_KHR_image	EGL_KHR_image_base
EGL_KHR_gl_texture_2D_image	EGL_KHR_gl_texture_cubemap_image
EGL_KHR_gl_renderbuffer_image	EGL_KHR_image_pixmap
EGL_KHR_lock_surface	EGL_KHR_fence_sync
EGL_KHR_reusable_sync	EGL_IMG_context_priority
FBDEV based EGL (optional)	

  - The FBDEV must be based on a triple buffer mechanism. It must operate based on page flipping instead of copying buffers.
  - To avoid a tearing problem, coordinate with LCD vsync signal. Generally, compositor enables vsync using the `eglSwapInterval()` API.

- DRI2-based EGL™ (mandatory)
  - Very important!
  - EGL™ must be implemented based on X11 DRI2 protocol.
  - EGL™ must support a texture from pixmap (using `EGL_KHR_image_pixmap` and `GL_OES_EGL_image`).

- GPU synchronization API
  - To avoid flickering, internal EGL™ API which waits for the completion of the buffer copy command is required.

- Header and Library name
  - Header files and libraries must be provided within the driver according to the Khronos API Implementers Guide.

Note: You may need to implement some memory management techniques for sharing memory between your CPU and the graphics processing unit.

Window System (Xserver)

The following figure illustrates the main parts of XServer and the counter parts in the Linux kernel.

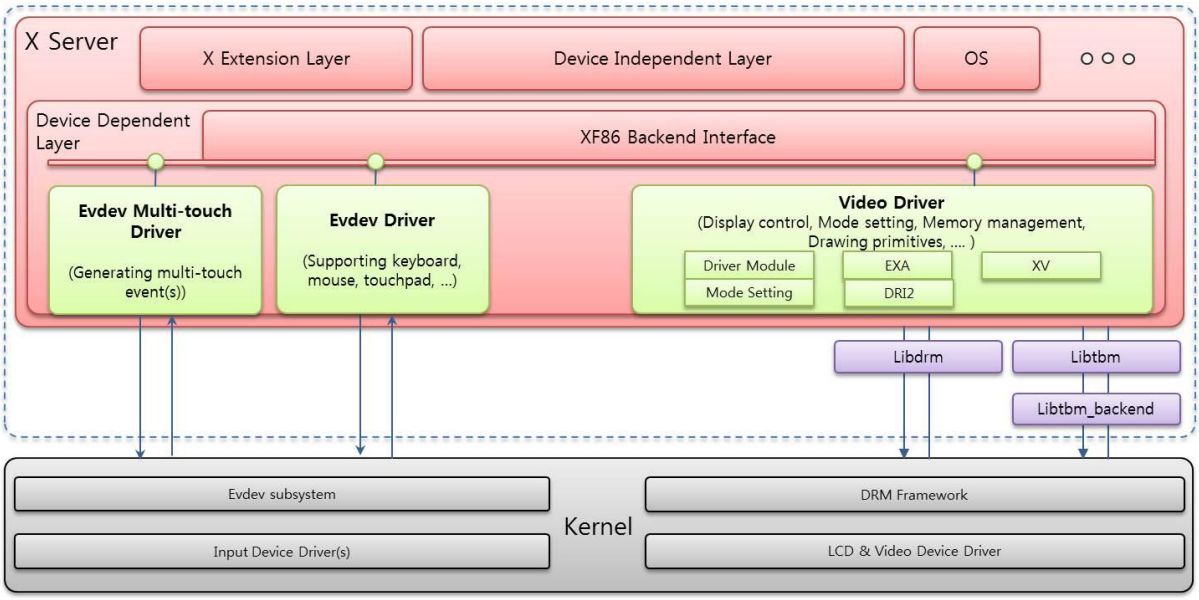

Figure 9. XServer components

Like other Unix/Linux systems, Tizen contains an XServer-based window system. The X window system provides an interface for manipulating resources, such as windows, pixmap, and gc.

X clients send requests to the X server for drawing something or manipulating windows. The X server accepts the client request and sends back a reply. In addition, the X server sends X events to the client when input events from devices are pending or when the X server encounters events that the X client is interested in, such as configure or exposure events.

Tizen uses X.org realization Xserver, which is an open source implementation of the X Window System (http://www.x.org/).

X.org server can be divided into 2 parts:

- DIX layer is a device-independent layer that performs device-independent tasks.
- DDX layer is a device-dependent layer that performs device-dependent tasks.

To make the X window system work on a device, you must port the X input/video driver in the DDX layer.

X Input Driver

There are 2 X input drivers: X Evdev driver and X Evdev-multitouch driver.

Each driver reads an input event stream from each input device node, makes X internal events, and puts them into the X server's internal event queue. For doing this, the X input drivers mainly use interfaces implemented on the xf86 DDX layer inside the X server.

Evdev driver:

- This driver interprets events from the kernel evdev subsystem for devices, such as a key(board) device, mouse device, or touchpad.
  Note: Basically, this driver can be used without any modifications.

- For information on the data structures and basic functions to be implemented for a device, see http://www.x.org/wiki/Development/Documentation/XorgInputHOWTO/.

Evdev-multitouch driver:

- This driver interprets events from the kernel evdev subsystem for touch-screen devices.

- It supports 2 kinds of MT protocol:

  - MT protocol A (which includes Tracking ID)

  - MT protocol B

Note: Basically, if the MT protocol B is supported by the kernel of a device, this driver can be used.

- You can support a legacy single-touch protocol by modifying the source code. Using the MT protocol B is recommended.

- To support a new touch protocol, do one of following:

  - Implement something for the new touch protocol inside the driver.

  - Clone the evdev multi-touch driver and implement the new touch protocol in the clone.

  - Convert from the new protocol to an existing protocol, which is supported by the evdev multi-touch driver by default.

X input driver configuration:

- The following options under “ServerFlags” are essential:

  - AllowEmptyInput: "true" means that the X server is started without any keyboard/mouse driver.

  - AutoAddDevices: "false" means that hot plugged input devices are not supported.

  - AutoEnableDevices: "true" means that a 'DevicePresenceNotify’ event is sent to all X clients by default.

- For more detailed information on the X configuration, see http://www.x.org/releases/current/doc/man/man5/xorg.conf.5.xhtml.
  By applying and modifying the configuration files (such as `xorg.conf` or files under the `xorg.conf.d` directory), you can apply optional features.

X Video Driver

The X video driver is a driver library that contains the implementation to support the xf86 DDX layer, to display something on the screen:

- Driver Module
  - Implementation of a driver module for the xf86 DDX.
- Mode Setting
  - Implementation for the devices to display images on it.
  - Kernel can support the mode setting (for example, Drm mode setting).
- EXA
  - Implementation for the graphic acceleration architecture. Memory allocation and extra draw primitives.
- DRI2
  - Implementation for DRI2 (Direct Rendering Infrastructure ver.2).
  - For the graphic acceleration, the graphics memory must be allocated and shared through the global memory management.
- XV
  - XFree86 offers the X Video Extension (XV), which allows clients to treat video as any other primitive and put video into drawables mode. By default, the extension reports no video adaptors as being available because the DDX layer has not been initialized.

Porting X Video Driver

The X video driver needs to implement the driver module and extensions mentioned above, using the interface provided by the xf86 DDX and X extensions in the X server. All steps are described in http://www.x.org/releases/X11R7.6/doc/xorg-server/DESIGN.txt.

3.4 Multimedia

This section describes the components of the Multimedia Framework.

3.4.1 Codec

Tizen Multimedia Framework provides HW codecs based on Gst-OpenMax IL Codec plug-in. The Gst-openmax IL codec plugin is a GStreamer plug-in package that allows communication with OpenMAX IL components.

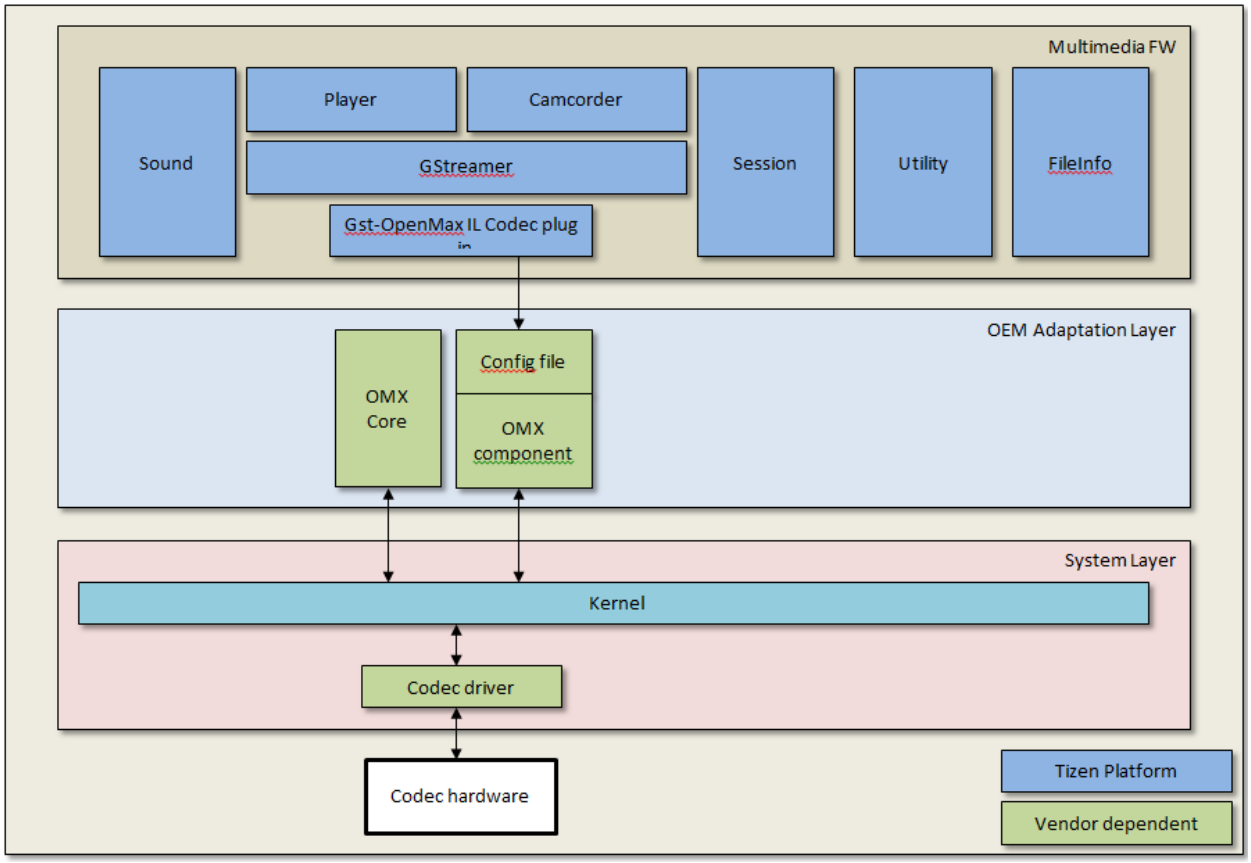

Figure 10. Codecs in the Multimedia Framework

Porting the OAL Interface

The Gst-openmax IL codec plugin is used as other GStreamer plugins. For more information, see http://www.freedesktop.org/wiki/GstOpenMAX.

References

For more information about OpenMAX IL components, see:

- http://www.khronos.org/openmax/il/

3.4.2 Camcorder (Including Camera)

The Multimedia camcorder framework controls the camera plugin of GStreamer to capture camera data from the device.

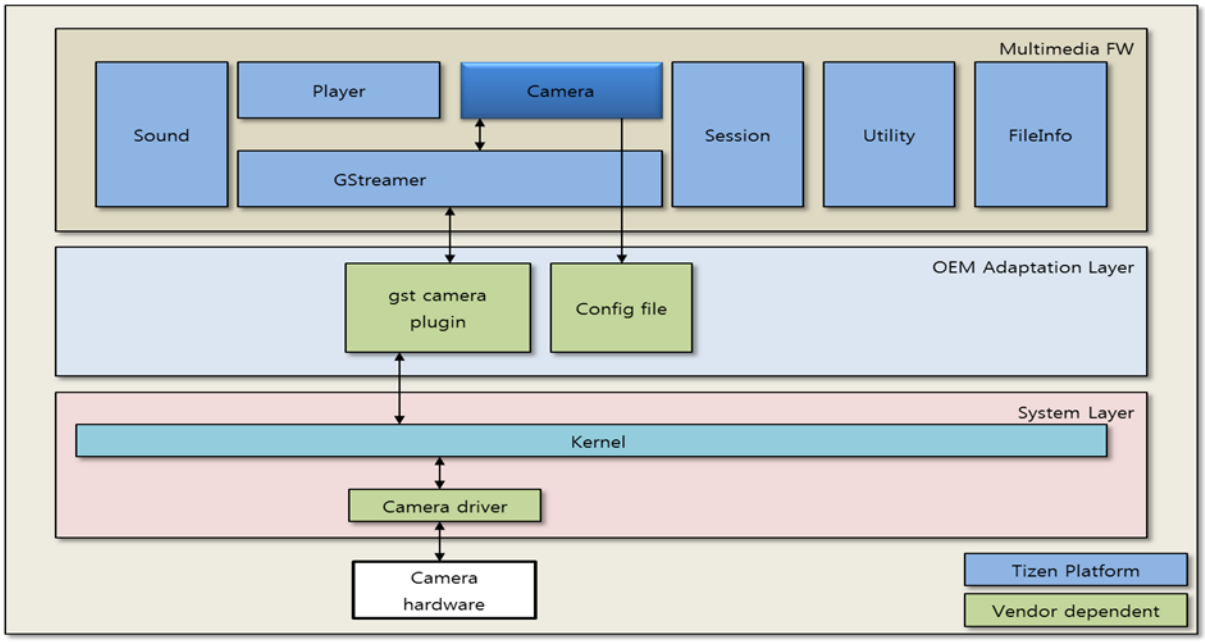

Figure 11. Camera in the Multimedia Framework

Note that the kernel interfaces controlling the camera device can differ for different chipsets, so the camera plugin must be implemented specifically for each chipset. Each config file contains its own specific hardware-dependent information.

- Camera source plugin for GStreamer
  - Gets camera data (preview or captured image) from the camera device
  - Inherits “GstPushSrc”
- Config files for the Multimedia Camcorder framework
  - `mmfw_camcorder.ini`: The camcorder settings file.
  - `mmfw_camcorder_dev_video_pri.ini`: This file includes the settings of the high-resolution rear camera.
  - `mmfw_camcorder_dev_video_sec.ini`: This file includes the settings of the low-resolution front camera.

Porting the OAL Interface

GStreamer Camera plugin:

- Because the kernel interfaces controlling the camera device can differ for different chipsets, the camera plugin must be implemented specifically for each chipset. To develop a new plugin, see the GSteamer plugin development guide (http://gstreamer.freedesktop.org/data/doc/gstreamer/head/pwg/html/index.html).

- The major functionalities for the camera are provided by the GStreamer camera control interfaces and signal callback, which are added by Tizen. The camera plugin must support them (`gst-plugins-base0.10/gst-libs/gst/interfaces/cameracontrol.h` interfaces and signal callback: “still-capture”).

- Any third party developer who wants to implement a camera source plugin must derive it from GstPushSrc.

References

For all GStreamer documentation, see:

- http://gstreamer.freedesktop.org/documentation/

To develop GStreamer plugins, see:

- http://gstreamer.freedesktop.org/data/doc/gstreamer/head/pwg/html/index.html

3.4.3 Radio

The radio interface part of the Multimedia Framework supports APIs to implement the following FM radio features, using Linux V4L2 interfaces:

- Tune a frequency
- Get and set a frequency
- Scan all available frequencies
- Seek up and down
- Get frequency signal

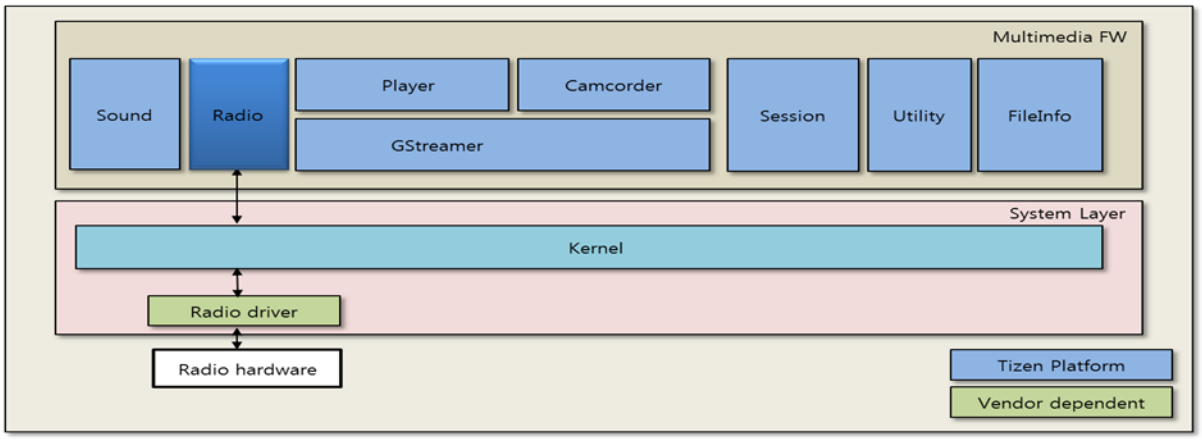

Figure 12. Radio in the Multimedia Framework

Porting the OAL Interface

The OAL interface for the FM radio is the Linux kernel V4L2 interfaces. The radio module directly uses the V4L2 ioctls to perform various radio hardware configurations.

References

V4L2 specification:

- http://v4l2spec.bytesex.org/spec-single/v4l2.html

3.4.4 Audio

The following figure illustrates the Multimedia Framework Audio components.

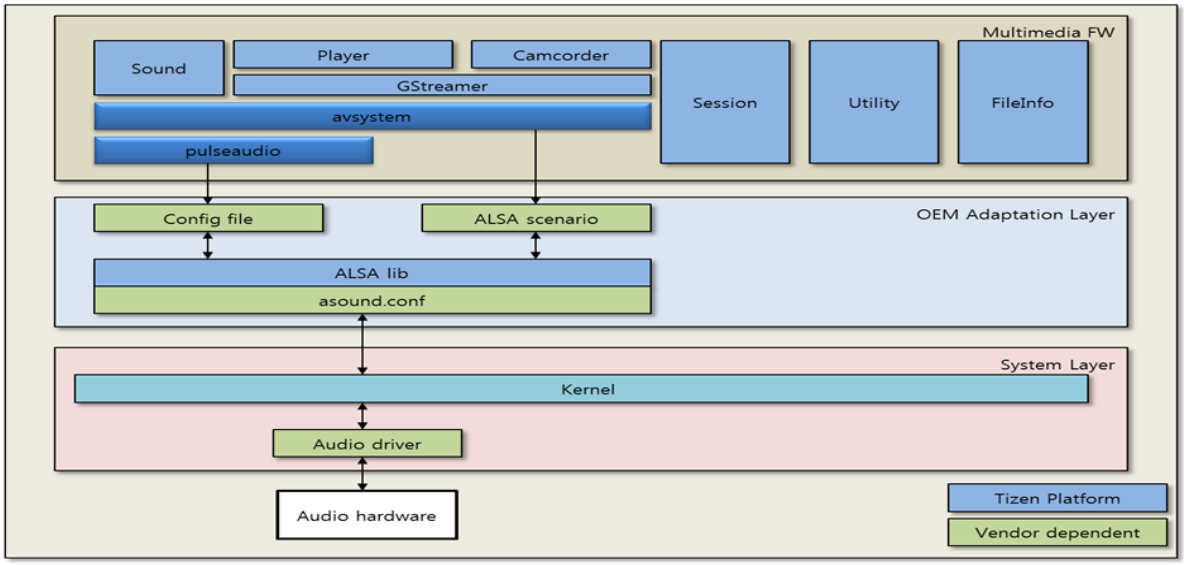

Figure 13. Audio in the Multimedia Framework

- Avsystem
  - Avsystem is a native Tizen component providing an API for handling the stream and the sound path in the upper layer.
- PulseAudio
  - PulseAudio is a sound server accepting sound input from one or more sources and redirecting it to one or more sinks.
- `asound.conf`
  - A configuration file for extra ALSA DAIs (Digital Audio Interface), such as AIF1 and AIF2.

Porting the OAL Interface

PulseAudio:

- PulseAudio in Tizen does not support udev.
- PulseAudio must be started as a system mode.
- To support a variety of devices, some configuration files have been separated from PulseAudio to mmfw-sysconf. For more information, see the configuration details below.
- Do not change the default sink in `/etc/pulse/system.pa`. The default sink is used by some modules. If it is changed, sound playback experiences problems:
  `load-module module-remap-sink sink_name=mono_alsa master=alsa_output.0.analog-stereo channels=1`

`asound.conf`:

- The name AIF2, AIF3 must not be changed.
- `/etc/asound.conf` has a dependency on the hardware. For more detailed information, see the configuration details below.

Configuration

To support a variety of devices, some configuration files have been separated from PulseAudio to mmfw-sysconf. PulseAudio has the following configuration files:

- `/etc/pulse/client.conf`
- `/etc/pulse/daemon.conf`
- `/etc/pulse/default.pa`
- `/etc/pulse/system.pa`
- `/usr/share/pulseaudio/alsa-mixer/profile-sets/default.conf`
- In addition, there are a few more configuration files. Currently, they do not have a major impact, but can be modified, if necessary.
  - `usr/share/pulseaudio/alsa-mixer/profile-sets/`
  - `usr/share/pulseaudio/alsa-mixer-paths/`

References

PulseAudio / ALSA

- http://www.freedesktop.org/wiki/Software/PulseAudio
- http://www.alsa-project.org/

3.5 Connectivity

This section describes the components of the Connectivity module.

3.5.1 WLAN

The main WLAN features are:

- WLAN (802.11 b/g/n)
- WPS PBC
- EAP (AKA, SIM, PEAP, TTLS)

The following figure illustrates the Tizen WLAN architecture.

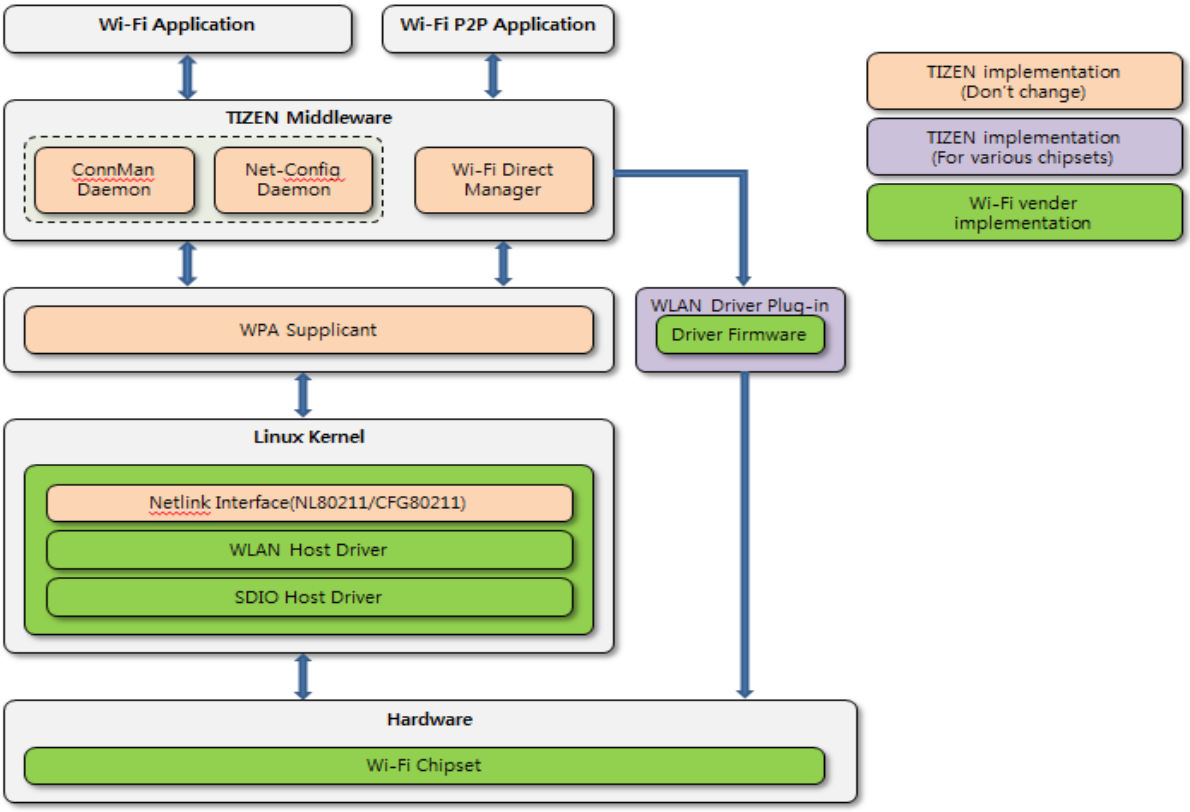

Figure 14. WLAN architecture

The WLAN architecture is centered on the Linux wireless (IEEE-802.11) subsystem. The Linux wireless SW stack defines the WLAN HW adaptation SW interfaces that need to be used in Tizen.

Connection Manager (ConnMan) is a daemon for managing internet connections within embedded devices running the Linux operating system.

WPA supplicant (wpa_supplicant) provides support for WPA and WPA2 (IEEE 802.11i / RSN). The supplicant is the IEEE 802.1X/WPA component that is used in the client stations.

Porting the OAL Interface

The WLAN driver plugin is specific to a Wi-Fi chipset. Wi-Fi chipset firmware and tools files must be copied to the WLAN driver plugin directory, built, and installed before testing the Wi-Fi functionality.

The Wi-Fi driver must create the `/opt/etc/.mac.info` file, which has the MAC address of the device.

The WLAN driver plugin contains a file called `wlan.sh` (`/usr/bin/wlan.sh`), which is used to load or unload Wi-Fi driver firmware. All other Wi-Fi-related functionality is handled by the ConnMan daemon.

References

- Connection Manager (ConnMan) project website: http://connman.net/
- Linux wireless (IEEE-802.11) subsystem: http://linuxwireless.org/
- Information on Linux WPA/WPA2/IEEE 802.1X Supplicant: http://hostap.epitest.fi/wpa_supplicant/
- Latest ConnMan release: http://git.kernel.org/?p=network/connman/connman.git;a=summary
- WLAN driver plugin git path: `adaptation/devices/wlandrv-plugin-xxx`

Reference kernel configurations

The following options must be enabled if the driver supports the cfg802.11 configuration API:

- CONFIG_CFG80211
- CONFIG_LIB80211
- CONFIG_MAC80211 (Enable this flag, if the driver supports the softMAC feature.)

The following options must be enabled if the driver supports wireless extension APIs:

- CONFIG_WIRELESS_EXT=y
- CONFIG_WEXT_CORE=y
- CONFIG_WEXT_PROC=y
- CONFIG_WEXT_PRIV=y
- CONFIG_WEXT_SPY=y
- CONFIG_WIRELESS_EXT_SYSFS=y

3.5.2 Bluetooth

Tizen uses open source Bluetooth components, such as Bluez and Obexd. Bluez and Obexd run as the daemon, and a Bluetooth Framework interface library is used by applications to access Bluez or Obexd over the D-Bus interface.

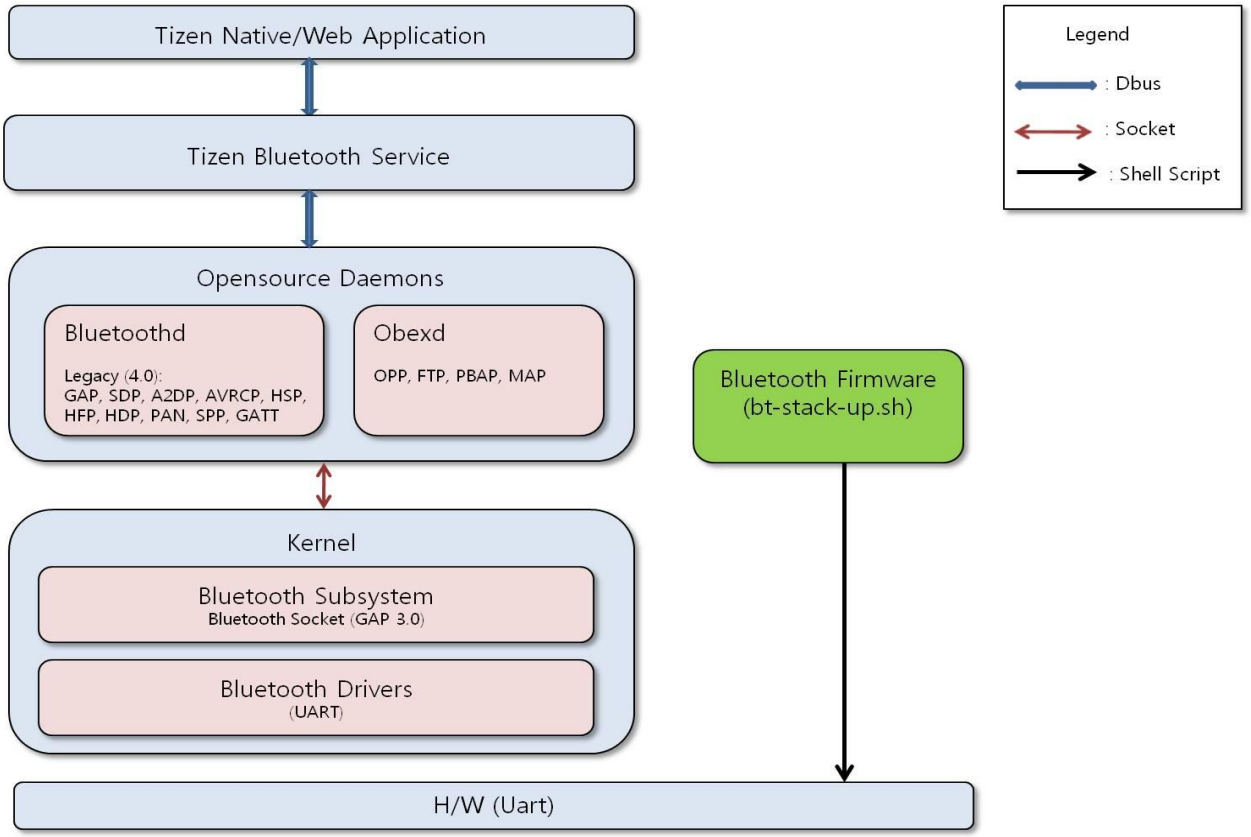

Figure 15. Bluetooth architecture

- Supported profiles: GATT, FTP, OPP, MAP, PBAP, A2DP, AVRCP, HSP/HFP, RFCOMM, HID, HDP, and PAN
- Application Definition: Provides a dialogue for the user. Controls BlueZ/ObexD/PulseAudio Daemon.

The Bluetooth Low Energy function is implemented in bluez&bluetooth-frwk. However, there are no SDK APIs about the BLE functions. The current plan is to include them in Tizen version 3.0.

Porting the OAL Interface

The following OAL scripts are run during the BT stack start and end sequences. These scripts invoke BT-chip-specific (such as Broadcom) scripts, provided by the chipset vendor to perform chip-specific configuration. The scripts are available with bluetooth-dev-tools.under. When the package is installed, it copies the scripts under `/usr/etc/Bluetooth/`.

- `bt-stack-up.sh`: This script file is used to run the hardware-specific script files to power up or start BT hardware along with the background processes, such as bluez and obexd.

- `bt-stack-down.sh`: This script file is used to run the hardware-specific script files to power down or stop BT hardware along with the background processes, such as bluez and obexd.

- `bt-reset-env.sh`: This script file is used to reset the BT chip by running `bt-stack-down.sh` along with the resource clean up.

Configuration

Some configuration changes must be made to enable the specific chipset, scripts, and other configuration information, such as UART speed and UART terminal (tty) to be opened specific to the chipset. The chipset vendor must provide the needed changes.

The following set-up is a configuration example of the BCM4330 Bluetooth chipset by Broadcomm:

- hciattach
  Project `bluez/tools/hciattach.c` is patched to enable the BCM4330 chipset-specific hciattach tool. This service attaches the BT UART HCI interface to the BT stack at baud rate of 3000000. It is also responsible for loading the BT firmware on BCM4330.

- The Bluetooth UART used is `/dev/ttySAC0`

- The Broadcom firmware used is BCM4330B1_002.001.003.0221.0265.hcd

- The UART speed configuration is 3000000 for BCM4330B1

- The bcmtool used is bcmtool_4330b1

- The .bd_addr contains the unique Bluetooth address, which is generated during the first Bluetooth activation

- To register the Bluetooth device:

```
bcmtool_4330b1 /dev/ttySAC0 –FILE=BCM4330B1_002.001.003.0221.0265.hcd –
BAUD=3000000 –ADDR=/csa/bluetooth/.bd_addr –SETSCO=0,0,0,0,0,0,0,3,3,0 –LP
```

- To attach a serial device using UART HCI to the Bluetooth stack for a broadcom device:

```
hciattach /dev/ttySAC0 –S 3000000 bcm2035 3000000 flow
```

- To run the Bluetooth daemon version 4.101:

```
bluetoothd
```

- To start up the device, set the device name, and enable the SSP mode:

```
hciconfig hci0 up
hciconfig hci0 name “Tizen”
hciconfig hci0 sspmode 1
```

- To switch on the Bluetooth radio:

```
rfkill unblock bluetooth
```

- To switch off the Bluetooth radio:

```
rfkill block bluetooth
```

3.5.3 NFC

The following figure illustrates the NFC components.

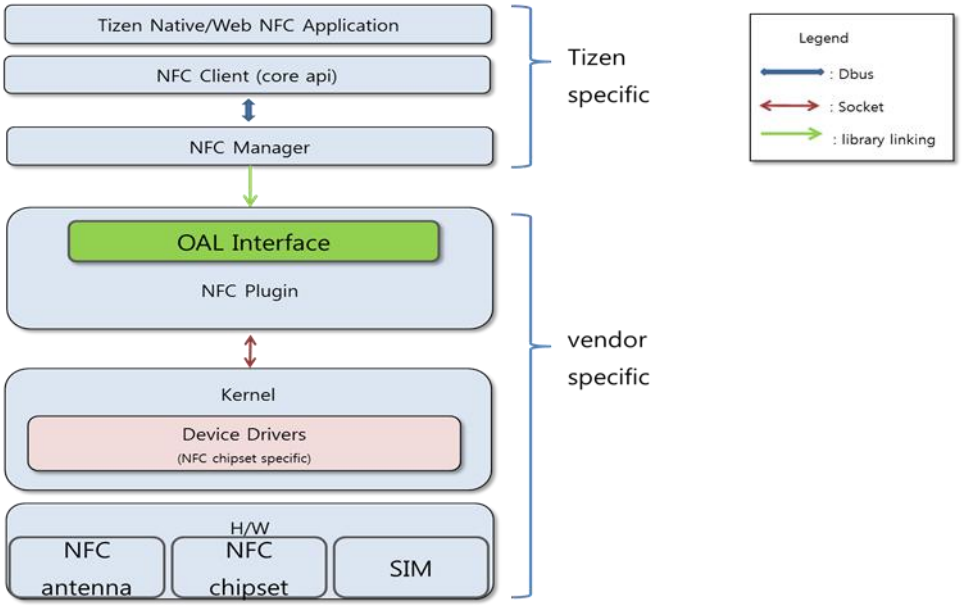

Figure 16. NFC architecture

- The NFC application enables the user to read and import the content written on an NFC tag, and write and save data in an NFC tag.
- The NFC client acts as an interface between the NFC application and the NFC manager, while writing or editing tag information in any physical tag.
- The NFC manager is the main interface, which actually deals with NFC physical tags, creates a connection with tags and detects it. It is a daemon process to control the NFC chipset (such as nxp pn544 or lsi). It provides the tag read/write service and basic P2P communication service. It provides a basic API to the client application.
- The NFC plugin and device driver act as an interface between the NFC chipset and the NFC Manager.

If you are chip vendor (for example, nxp, Broadcom, or lsi), you must provide the NFC chipset, device driver, and NFC plugin for the Tizen NFC Framework. For a reference implementation of the NFC plugin, go to the Tizen public code site and download adaptation/devices/nfc-plugin-nxp.git.

To use any specific NFC chipset in the Tizen NFC Framework, you must create the OAL interface in the NFC plugin provided by the chip vendor.

Porting the OAL Interface

The NFC plugin is implemented as a shared library and it interfaces the Tizen nfc-manager and the vendor NFC chip. The NFC manager loads the `libnfc-plugin.so` library at runtime from `{library_path}/libnfc-plugin.so`. Any vendor-specific plugin must be installed in the same path. The plugin must be written with predefined OAL API interfaces.

During the initialization, the nfc-manager loads the `libnfc-plugin.so`, searches for the `onload` API, and calls it with an interface structure instance as an argument for mapping all the OAL interfaces.

The OAL/OEM interfaces are implemented according to the underlying NFC chipset. Once the mapping is done, the NFC manager interacts with nfc-plugin, which implements the vendor-specific OAL interfaces.

The `nfc_oem_interface_s` struct is exported in the nfc-plugin. Using this interface structure, the nfc-manager communicates with the OAL interfaces at runtime.

For a reference implementation of the OAL interfaces, go to the Tizen public code site and download adaptation/devices/nfc-plugin-nxp.git, and see src/oem/oem_nxp.c.

References

- NFC Technical specification

3.6 Telephony

This section describes the Telephony module.

Telephony Architecture

Telephony supports the plug-in architecture, which provides flexibility to include various types of predefined plug-ins into the system with very few changes.

The following figure illustrates the Telephony plug-in architecture. Function calls across various plug-ins go through the Tizen Telephony library (libtcore) APIs.

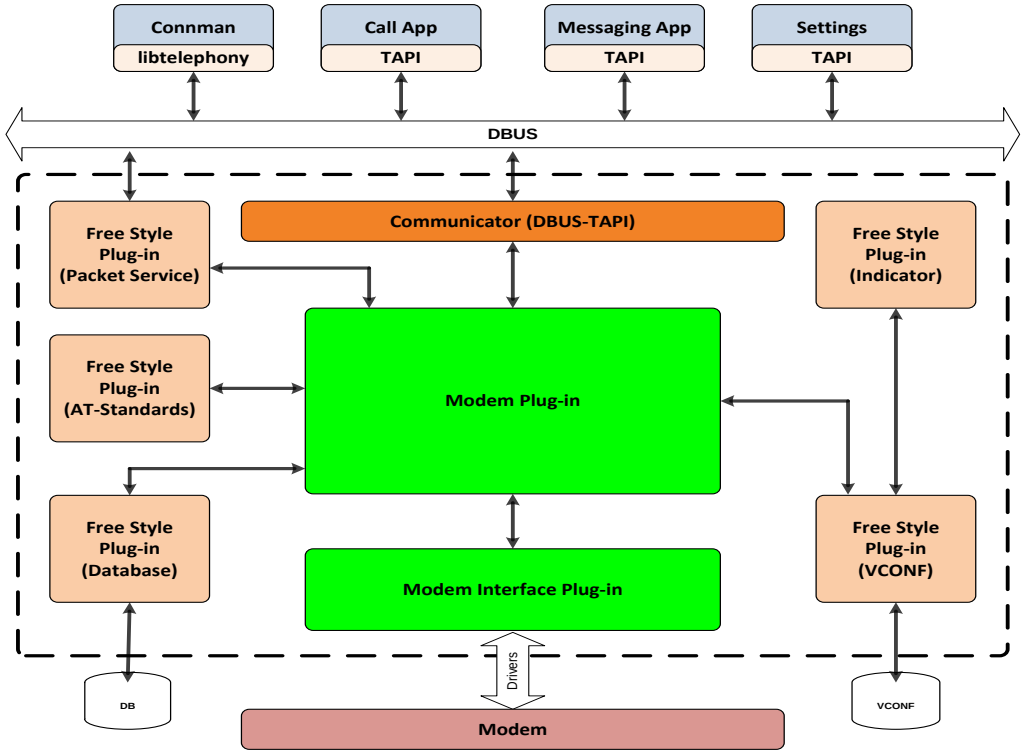

Figure 17. Telephony architecture

Telephony libraries

Telephony provides 2 libraries:

- Telephony API (TAPI) library
  The TAPI library (or simply TAPI) is a standardized interface provided to applications to interact with Tizen Telephony.

- Core Telephony library (or libtcore)
  The Core Telephony library (or libtcore) provides an API framework for Tizen Telephony to interwork. The libtcore provides APIs for:

  - Creation, destruction, and maintenance of various server components, such as Server, Communicator, HAL, Plug-in, and Core Object
  - Storage maintenance, queue mechanism, and general utility
  - CMUX support (creation, destruction, and processing)
  - AT parser

Telephony Plug-ins

The following Telephony plug-ins are available:

- Communicator plug-ins
  - DBUS communicator is provided for the interface between TAPI and the Telephony Server.
- Modem plug-ins
  - Contain core functional units providing Telephony functionality.
  - Maintain and manage Telephony states.
  - Maintain and manage the database related to Telephony.
- Modem interface plug-ins
  - Interface between the Telephony Server and the Communication Processor.
  - Hardware-specific plug-ins which define hardware capabilities and usage.
  - Modem interface plug-in is also called the “hardware abstraction layer” (HAL).
- Free style plug-ins
  - Provide completely independent functionality irrespective of hardware (Communication processor).
  - Include plug-ins, such as packetservice, storage, and indicator.

Porting OAL Interface

OEM vendors can port each and every available plug-in within Telephony according to their needs. It is not mandatory to port all the plug-ins to support a specific hardware. OEMs need to specifically implement the Modem and Modem interface plug-ins to support their hardware.

Any telephony plug-in must provide a plugin descriptor structure (see `/libtcore/include/plugin.h`). The descriptor structure of each plug-in must be named as ‘plugin_define_desc’. The server obtains the address of this symbol in order to provide control to the plug-in to execute its defined functionality. The initialization order of a plug-in among various other Telephony plug-ins is defined based on the plug-in’s ‘priority’.

All Telephony plug-ins must be installed in the following folders:

- Modem plug-ins: `/usr/lib/telephony/plugins/modems/`
- Other plug-ins: `/usr/lib/telephony/plugins/`

3.7 Security

This section describes the Security module.

3.7.1 Access Control (Smack)

The following figure illustrates the Tizen security model.

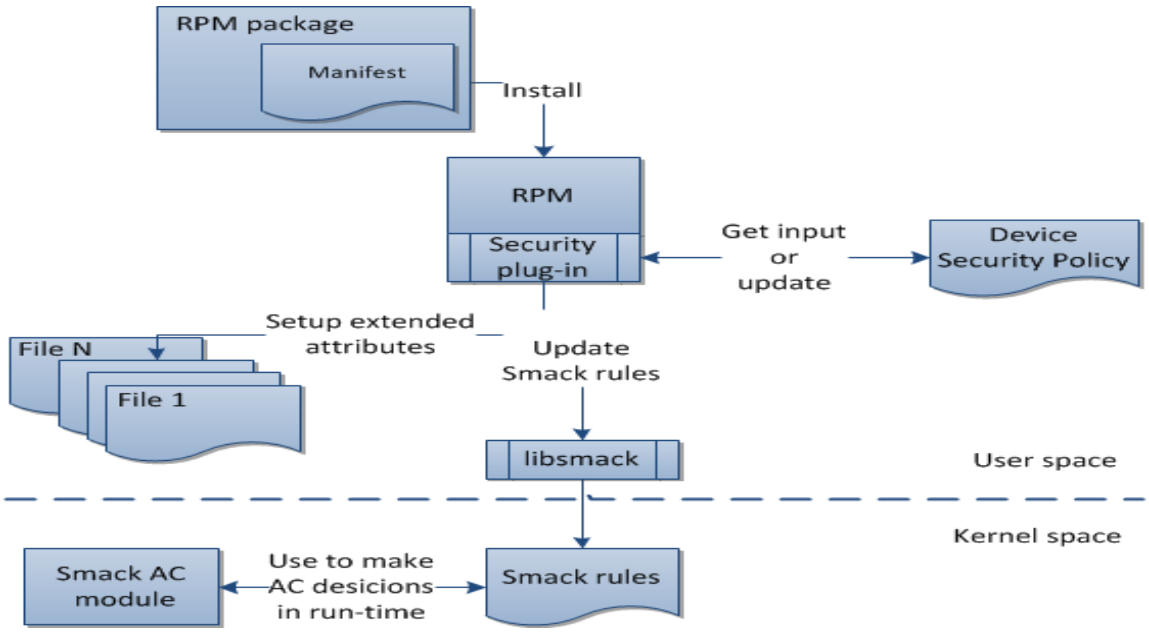

Figure 18. Security model

To achieve the security model, Tizen 2.3 uses:

- Discretionary Access Control (DAC): File System permissions and Access control list (ACL)
  http://en.wikipedia.org/wiki/Discretionary_Access_Control
- Mandatory Access Control (MAC) with the kernel LSM Smack
  http://en.wikipedia.org/wiki/Mandatory_access_control

SMACK in Tizen 2.3

Smack supports 2 primary security requirements:

- Control over privileged resources at the granularity of the application
- Application isolation with controlled sharing

Security Models based on Smack

- Privilege
  Services that are being used by applications must control if the caller has sufficient privileges to call

each API. In Tizen 2.3, this level of access control is done using very detailed Smack policy on IPC mechanisms. In other words, privilege is composed of smack rules. You can confirm this in `/usr/share/privilege-control/*`.

  - For an overview of Tizen privileges, see: https://developer.tizen.org/dev-guide/2.2.1/org.tizen.gettingstarted/html/tizen_overview/privilege.htm

  - For a list of the Tizen privileges, see: https://www.tizen.org/ko/privilege?langredirect=1

- Manifest
  A Web or native application developer must describe a privilege they want to use in the manifest file. In addition, each RPM package must have a manifest file where developers can specify the access control domain in which their application must be running and potential additional security policies for the application.

- Installing applications
  The installer uses the content of the manifest file to set the security context of the installed application or RPM package. The installer uses the libprivilege-control component to handle the Smack security database.
  The installer updates the database of the application's smack rules in `/opt/dbspace/.rules-db.db3`. The smack rules for the RPM package are stored in `/etc/smack/accesses.d/*`.

- Smack auditing
  By default, all denied events are audited. Kernel-space denied events remain on dmesg. Also, any user-space denied event on IPC mechanisms is recorded on dlogutil.

Reference

For smack, see:

http://en.wikipedia.org/wiki/Smack_(Linux_security_module)

For manifest and installing applications, see:

https://wiki.tizen.org/wiki/Security/WebApps_and_Smack#Manifest_for_WebApps

For manifest and installing the RPM package (Platform Module), see:

https://wiki.tizen.org/wiki/Security/Application_installation_and_Manifest#Manifest_file

3.7.2 Certificate Management

In the Public Key Infrastructure (PKI) scheme, certificates are used to prove the ownership of public keys. Because Tizen uses the public key scheme, Cert-svc (Certification service) and Cert-svc-vcore are implemented to manage certificates efficiently and only allow the installation of applications with valid signatures.

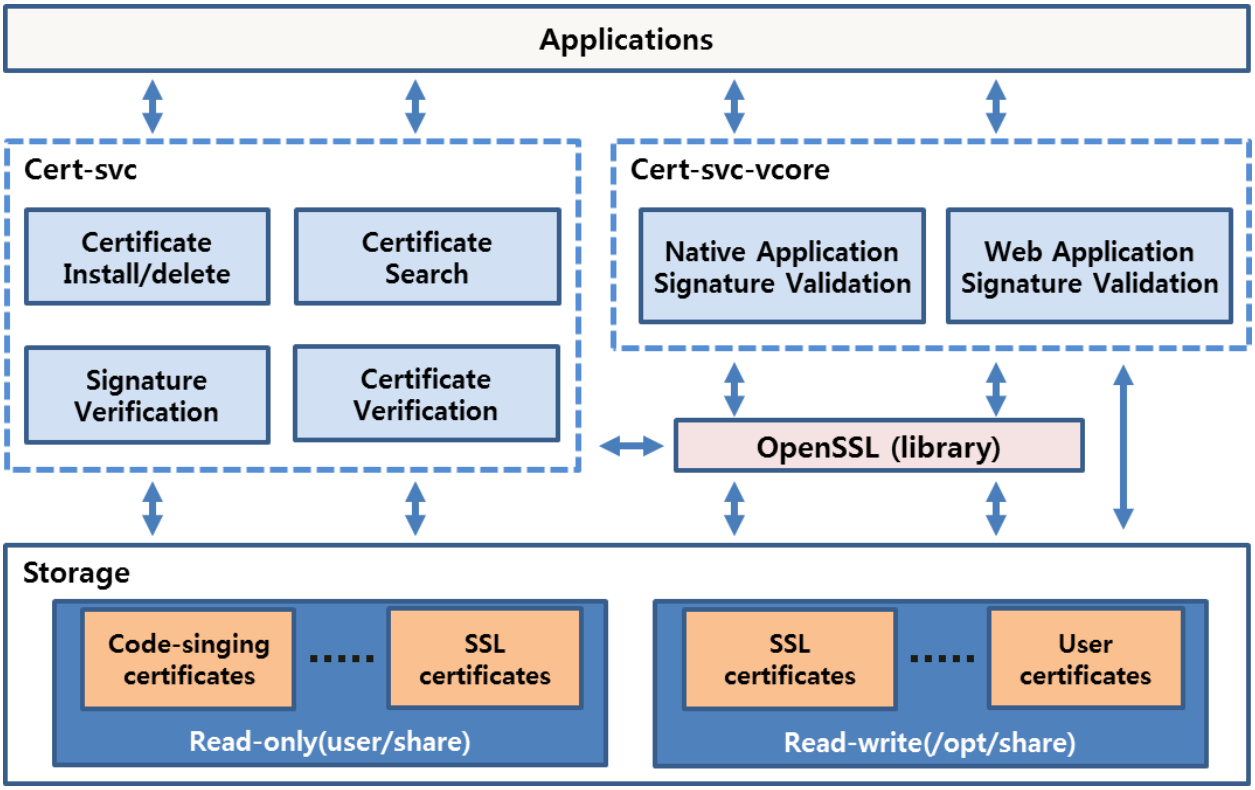

Figure 19. Certificate management

The following table lists the main functions of the certificate components.

Table 2. Main functions

	Function	Description
Cert-svc	Certificate Installation	Install certificates to the appointed location.
	Certificate Deletion	Delete unnecessary certificates.
	Certificate Information Extraction	Extract information from X.509 structure certificates.
	Certificate Search	Search certificates using user request information.
	Certificate Verification	Check whether certificate validation is normally issued.
	Signature Verification	Verify a signature using a message, signature, and certificates.
Cert-svc-vcore	Native App Signature Verification	Check the validation of native application signatures.
	Web App Signature Verification	Check the validation of Web application signatures.

Reference

- Certification Service Programming Guide V 0.1

3.7.3 Anti-Virus (CSR Framework)

CSR (Content Screening and Reputation) Framework consists of the content security framework and security plug-in. In Tizen, they are shared libraries.

- Content screening:
  Enables the caller module and applications to scan content (data, file).

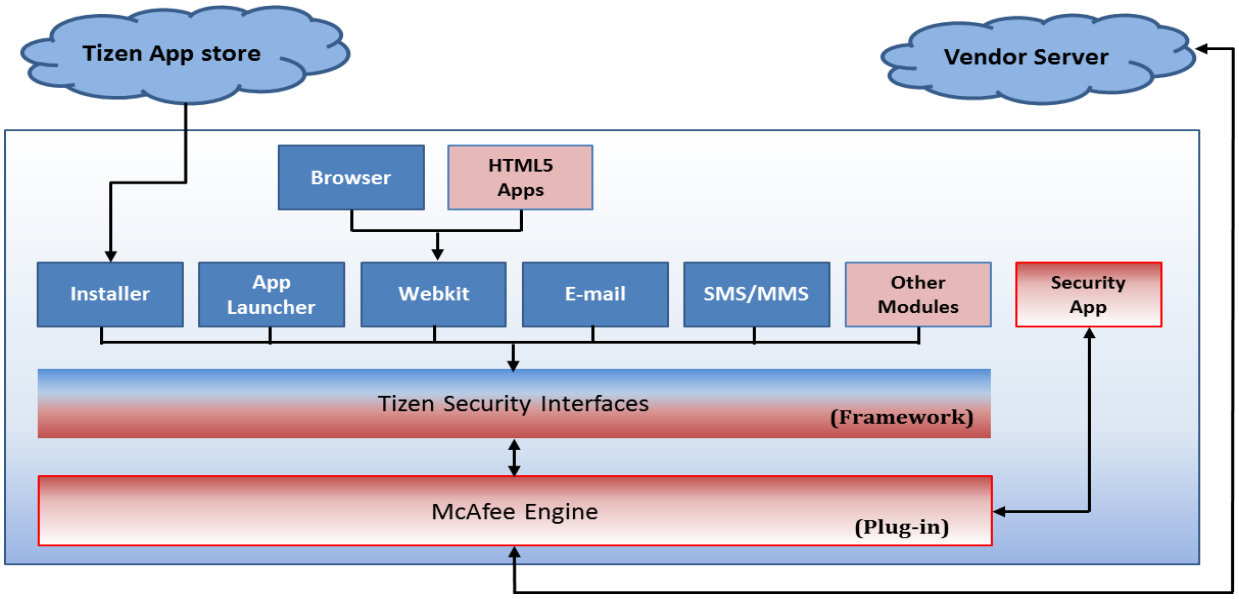

Figure 20. Content screening

- Reputation (Web Protecting):
  Can protect the user around the world from Web-based malware threats, browser exploits, and identity theft.

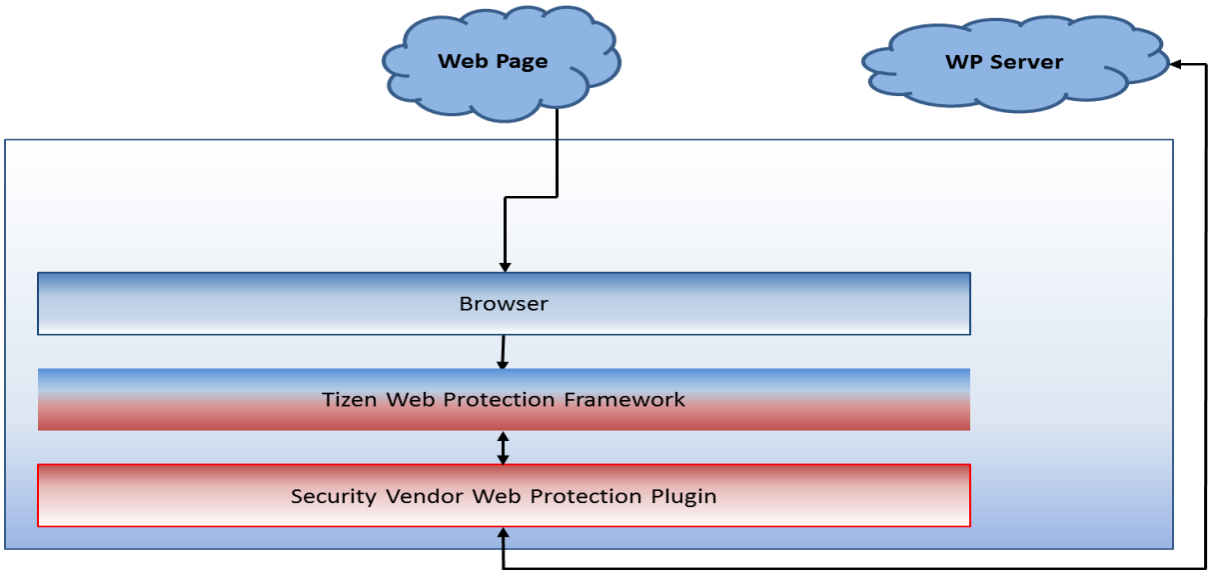

Figure 21. Reputation

Content Security Framework

- Shared library: libsecfw.so
  This file is linked directly to the system component which invokes the security API. It is responsible for plug-in (engine) loading and reloading.
  Always try to load the new plug-in from the `/opt/usr/share/sec_plugin/` path.

Security Plug-in

- Shared library: libengine.so (content screening), libwpengine.so (reputation)
  The security plug-in is loaded in runtime and installed along with the security application package.

Anti-virus App Package Format

- Package format must conform to the Tizen application format, which is a simple zip file.
- Package contains:
  - `/bin/…` (Application executables)
  - `/res/…` (Icons, pictures, and other resource files)
  - `/data/…` (Data for the application)
  - `/lib/plugin/libengine.so` or `libwpengine.so`
    (The installer copies this file to `/opt/usr/share/sec_plugin/libengine.so` or `libwpengine.so`.)
  - `/database/…`
    (Data for the engine. For example, Signature DB. This directory is readable to everybody.)
  - `/info/manifest.xml` (Contains the proper app type. For example, <category="sec-app"/>.)

Tizen Installer Enhancements

- Checks the signature/certificate and makes sure it is certified by a trusted party.
- Copies `/lib/plugin/libengine.so` to `/opt/usr/share/sec_plugin/libengine.so` or `libwpengine.so`.
- Copies all files to `/opt/usr/apps/[package id]/….`
- Sets the smack label of `/opt/usr/apps/[package id]/database` to"sec_database".
- All applications which use the content security framework already have rules to read "sec_database".
- Gives permission for the security application.
  (Privilege for the security check, like access to the file system or other applications.)

Reference

- Tizen Content Security Framework Proposal (Document version 1.0.2 2013, McAfee, Inc.)

3.8 Location

The Location module provides location-based services (LBS) including the position information, satellite information and GPS status.

The main features are:

- GPS (Global Positioning System)
- Getting the current position, last known position, accuracy, distance, and velocity
- Getting the satellite information of GPS and GLONASS
- Notifying the user when they enter or exit a predefined set of boundaries, known as geo-fence, such as school attendance zones or neighborhood boundaries.

Location Framework

The following figure illustrates the Location framework.

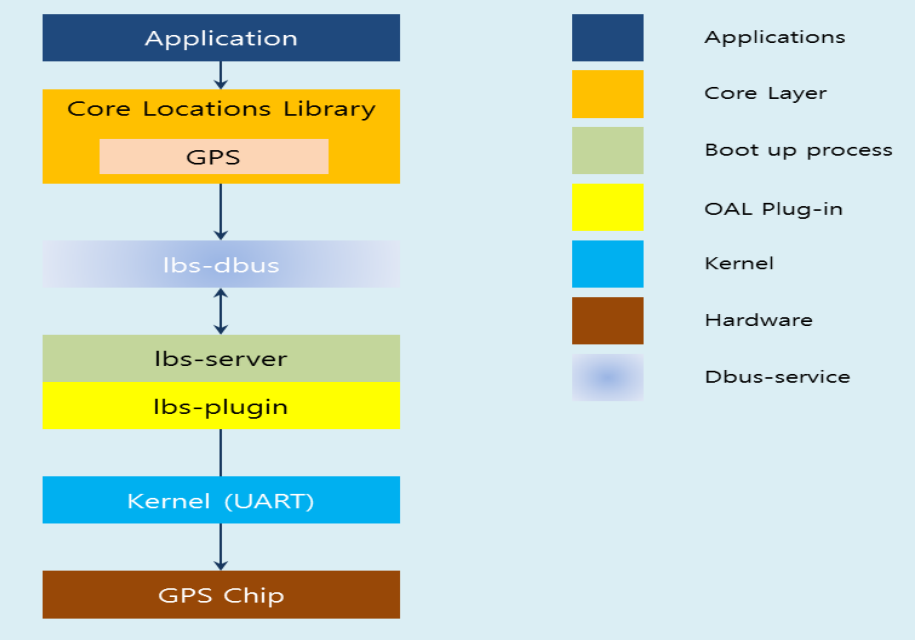

Figure 22. Location framework

- Location Library:
  - Contains the location providers that can be used by the native locations to get the services.
  - GPS provides position information, velocity, and satellite information. It is used to get the current position of a device.

- Lbs dbus:
    - This is the IPC used to communicate between the Location module and the GPS Manager daemon.
- LBS Server:
    - Provides position, velocity, NMEA, and satellite information by communicating with a GPS chip.
    - Functionalities of the LBS Server:
      - GPS initialization/de-initialization and open/close control.
      - Provides the position result for the location library.
      - Location session management-determination for session termination based on session status.
      - Serial interface with the GPS receiver.
      - Enables the GPS chipset to support standalone GPS positioning methods.
      - Supports the standalone operation mode, where a GPS receiver device in which the receiver provides all of its own data needs and performs all position calculations, without connection to an external network or server.

Porting the OAL Interface

The LBS plugin is implemented based on the Tizen LBS server for a vendor-specific GPS device.

The LBS plugin is implemented as a shared library and the lbs-server loads a specific LBS plugin at runtime. An LBS plugin must be written with predefined interfaces. The lbs-server-plugin-dev source package is installed on OBS by adding the following command in the package spec file:

- `BuildRequires: pkgconfig(lbs-server-plugin)`

Within the lbs-server-plugin-dev package, the source files are located in:

- `/usr/include/lbs-server-plugin/*.h`
- `/usr/lib/pkgconfig/lbs-server-plugin.pc`

The `gps_plugin_intf.h` file includes the API interfaces for the communication between the lbs-server and its GPS plugin:

```
typedef struct {
   /** Initialize plugin module and register callback for event delivery */
   int (*init) (gps_event_cb gps_event_cb, gps_server_param_t * gps_params);
   /** Deinitialize plugin module */
   int (*deinit) (gps_failure_reason_t *reason_code);
   /** Request specific action to plugin module */
   int (*request) (gps_action_t gps_action, void *data, gps_failure_reason_t *reason_code);
} gps_plugin_interface;

const gps_plugin_interface *get_gps_plugin_interface();
```

`get_gps_plugin_interface()` must be exported in the LBS plugin. It gives the gps_plugin_interface structure to the lbs-server, and the lbs-server communicates through these interfaces. When the gps-manager is started, the GPS plugin is loaded and the `init()` function is called. At this moment, a GPS device must be initialized (for example, for power control and firmware download).

4. Optimization

Usually, 2 main approaches are used to carry out optimization tasks. The first one is static (source code) analysis, which is usually helpful only in rather simple applications. Static analysis is usually difficult to apply in complex software, which provides, for example, many services with different execution threads, system IPC usage, and third-party module (plug-in) support. In that case, you must analyze the application behavior at runtime with various usage scenarios, and the second approach, dynamic analysis, is more effective. The Tizen platform provides the necessary tools for dynamic application profiling.

The dynamic analysis process is split into several base stages:

- Profiling session preparation
- Data collection
- Visualization of the results

At the preparation stage, the developer selects the data that currently needs to be analyzed: points of interest (such as libraries or functions), features to profile (such as memory, file IO, and network), and other parameters (such as sampling period). All that data is collected on a target device during the profiling session, and sent back to the host system. The host parses and stores the received data, performs the needed analysis, and displays the results in a human-readable form. This approach allows the developer to observe how the program behaves in a complex environment and how different system aspects affect it. The following figure illustrates the overall profiling process for a Tizen application.

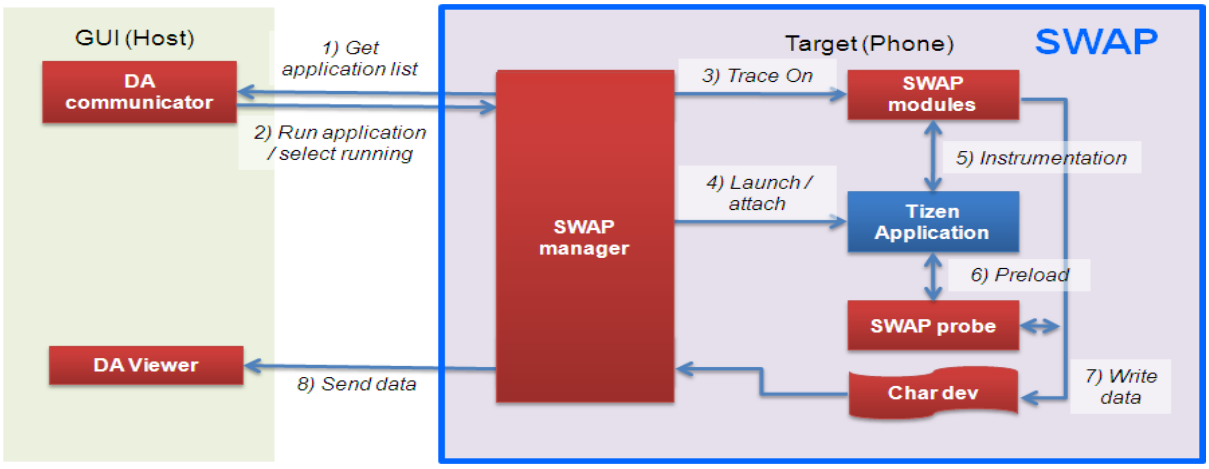

Figure 23. Profiling process

System Wide Analyzer of Performance (SWAP)

System Wide Analyzer of Performance (SWAP) is a dynamic binary instrumentation engine that allows you to carry out system-wide performance, memory, power consumption profiling, and optimization. It is based on a kprobe mechanism, and provides the facility to collect all the necessary data for analyzing the application and whole system behavior at runtime. SWAP is used as a backend for profiling data collection on a target device.

SWAP key features:

- Kernelspace profiling:
  - Can execute custom handlers when an instruction at a given address is hit.
  - Functions profiling: Execute custom handlers when a specified function is entered and exited (perform, for example, arguments or return value analysis, or timestamp recording).
  - Profiling of, for example, interrupt handlers, scheduler, drivers, syscalls, IPC, IO, graphics, and input events.
- Userspace profiling:
  - Functions instrumentation with custom entry/exit handlers.
  - Multithreaded and multiprocess application support.
  - Libraries instrumentation, including a system-wide mode in which the data is collected from all the running processes.
  - “Already running” profiling: no application/system restart required.
- Easy data extraction:
  - For example, function arguments (either kernel or userspace), task registers, and current system parameters.
- Multiarch support:
  - ARM, ARM64, x86, and MIPS.
- No source code or debug information needed.
- No program/kernel recompilation required.
- Modular architecture:
  - Easy to implement new SWAP-based tools for specific tasks (such as memory analysis, performance measurement, or power consumption estimation).
- Can be compiled as a separate module or built into the kernel image.

Dynamic Analyzer (DA)

Dynamic analyzer is a GUI frontend designed for performing application behavior analysis based on the collected data from the target device. It provides:

- Timeline chart: CPU load, memory, flash, network usage, UI events, application lifetime, and custom chart
- Summary: Failed API, resource leaks, functions profiling, and warnings
- Analysis components: File IO, Threads, UI, Network, OpenGL® ES, Scheduling, and System Info

- UX and other information: call traces, record and replay, source code link, range-based analysis, runtime screenshots, service/hybrid application support, platform library support, and multi-process support

DA key features:

- Find possible optimization points, such as bottlenecks and unnecessary waiting.
- Analyze the cause of the problem, such as leaks and functions usage bugs.
- Efficiency analysis: network, OpenGL® ES

Appendix A. Glossary

Table 3. Glossary

Term or abbreviation	Description
ALSA	Advanced Linux Sound Architecture
EAP	Extensible Authentication Protocol
Tizen	Linux-based open source operating system for devices (such as mobile, TV, and IVI)
V4L2	Video4Linux2
WLAN	Wireless Local Area Network
WPA	Wi-Fi Protected Access
WPS	Wi-Fi Protected Setup