Introduction

GENIVI Alliance delivers a set of documents and code modules that allow you to easily build your In Vehicle Infotainment solution.

**GENIVI Technical Deliverables**

- Demonstrators (GDP)
- Reference Architecture
  - Pre-integrated IVI Platform (Baselines)
  - Individual Software Components
  - Standard Interfaces & APIs
- Specification

These deliverables are detailed in following chapters (see menu on the left).

Several types of companies can take benefit of this Resource Kit.

If you are an Automaker or a Tier one supplier to Automakers, you can use it:

- To evaluate technologies,
- To build a platform basis for a serial product; your development will be much faster and cheaper thanks to the great amount of existing code maintained by the community.

If you are a Software Company you can use it:

- To demonstrate an integration of your software products in a GENIVI environment,
- To get the “Works with GENIVI” branding.
- To build a platform basis for a serial product that you intend to deliver to an Automaker or to a Tier one supplier.

If you are a Silicon Vendor or Hardware Company you can use it:

- To demonstrate that a GENIVI stack runs on your hardware,
- To demonstrate the performance of your hardware when running a GENIVI stack,
- To get your hardware promoted on GENIVI website.

If you are a journalist you can use it to tell the world how great GENIVI is! 😊
Different individual profiles will use the Resource Kit in different ways

We will consider two categories in the following chapters: managers – developers.

The following logo will indicate paragraphs that allow to go in deep details and will be of great interest for developers but that manager may prefer to skip.

IVI Development process using GENIVI

To begin your experience with all that GENIVI delivers, we will start with GENIVI Demo Platform (GDP) in the first chapter and then guide you along the details that will help you to build your IVI solution and customize it to your needs. This Resource Kit will help you to understand how a GENIVI "tool box" consisting of many different tools can bring efficiency in IVI development process.
1_GENIVI Demo Platform (GDP)

Rationale

GDP is a technology demonstrator that enables adopters to understand how GENIVI's software and standard interfaces work together in an executable demo. Because much of GENIVI's software is not user-facing, this demonstrator brings the GENIVI middleware to life by executing demo applications that use the various components in the middleware.

Value

A fully functioning GDP proves that GENIVI's component choices can work together and helps developers to understand how to use GENIVI software to integrate, test, and build their own IVI solution.

See the demo live

Before presenting GDP it is important to understand that GENIVI is not providing a fully configured solution but rather a tool box that you can use to build your own solution. Along your journey in GENIVI you will have to choose your path depending on your needs and preferences. And this is the case with the GDP in that there are several possible variants of GDP (a variant being a combination of target hardware, a software baseline, and demo applications).

One variant is presented in the following movie.

GDP variants

Other variants can be built. To configure your demo you have to choose from bottom to top a target Hardware, a GENIVI Baseline, and the Demo Apps you want to integrate.

<table>
<thead>
<tr>
<th>Demo Apps</th>
<th>Audio Manager Monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Web Browser PoC</td>
</tr>
<tr>
<td></td>
<td>3D Navigation rendering</td>
</tr>
<tr>
<td></td>
<td>Fuel Stop Advisor</td>
</tr>
</tbody>
</table>

| App&HMI Framework         | Qt                    |

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Yocto</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Baserock</td>
</tr>
<tr>
<td>Hardware</td>
<td>Intel Based Minnowboard MAX</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------</td>
</tr>
</tbody>
</table>

## Build your own demo

Prerequisites: one 64-bit Linux workstation (e.g. Ubuntu 14.04.3 LTS 64-bit) with:

- At least 50 GB available on disk
- High-speed Internet connectivity
- Git installed

The [Open Source Code](#) and [instructions](#) to build your demo can be found in GENIVI git repository and GDP Wiki pages.
2_Reference Architecture and Compliance Specification

Rationale

GENIVI builds its reusable, IVI platform based on a flexible but standard Reference Architecture. This Reference Architecture serves as a blueprint for building a full IVI solution and it also shows the primary target functional areas of GENIVI's work. The Compliance Specification steers the standardization, specifying exactly what is expected in the architecture implementation to build a GENIVI Compliant(r) solution.

Value

This common description allows you to use trusted open source components when building an Infotainment solution. The architecture also helps to avoid fragmentation on non-competitive areas of the software design. Members using the Compliance Specification to build a solution can register their product under GENIVI's compliance program and obtain a license to use the GENIVI Compliant(r) trademark in their solution marketing materials. This branding will give buyers confidence that the solution adheres to GENIVI's standards and to the component choices that ease integration and extension of the solution.

Architecture Overview

GENIVI does not deliver a given design but a toolbox that will let you make choices. In order to have a common understanding of the pieces in the box with a high level view of their role, a reference Block Diagram has been drawn.

GENIVI focus is on the “middleware” represented here by the yellow and the purple layers.

The yellow layer deals with key infrastructure domains. The smaller blocks map to individual Software components, that are available in the Open Source Community, some of them being developed in GENIVI Open Source Projects.

In the purple layer above, the smaller blocks represent functional domains that may need several Software components to be addressed, part of the Software Components needed may be commercial Software components.

GENIVI compliance specification

GENIVI compliance specification defines requirements on the middleware. The content of the Compliance Specification document can be seen as a list of Software components that are described in different ways depending on their importance in the system built and the level
of standardization that GENIVI wants to achieve in the area.

Thanks to the compliance specification mechanism GENIVI can steer standardization level to maximize reuse where it is worth and let freedom where competition brings more value. The use of P2 level lets some flexibility both on features content (i.e., navigation interfaces are P2 as they would not be needed in a radio) and on standard enforcement (where the community is not ready to decide a unique standard), the use of P1 level leads to a strict standardization of either code, interfaces or requirements.

Companies can show their commitment to the standard through GENIVI Compliance programs.

- "GENIVI Compliant" branding for platforms or products
- "Works with GENIVI" branding for pieces of code.

Learn more about these programs on [http://www.genivi.org/genivi-compliance-program](http://www.genivi.org/genivi-compliance-program)

**How does the compliance specification cover the architecture block diagram?**

GENIVI target has been defined and work is still in progress in several areas.

The detailed status is available in the documents listed in the next section. An overview on how GENIVI defines its roadmap is also given in the appendix 2 “GENIVI Roadmap”.

**More details on Architecture and Compliance Specification**

You can get a description of each domain of the block diagram in Appendix 3.

A static high level block diagram is not enough to describe an architecture. Developers need to understand deeper the data structures used and communication between the components.
3_GENIVI Baselines

Rationale

Architecture description and compliance specification are documents and not code. In order to be sure that what has been written in these documents really works, the first verification step is to confirm that one can build and run an In-Vehicle Infotainment software stack that complies with these documents.

Value

The Baselines prove the compliance specification consistency and can constitute a starting base to prototype and design a product. They are used in the GENIVI Demo Platform.

Description

What is a GENIVI baseline?

A GENIVI "baseline" is an integrated, installable image of GENIVI components and their dependencies based on a time-based snapshot of a version of the GENIVI compliance specification. A GENIVI baseline contains about 150 components, each of which performs different functionality. A GENIVI baseline is not to be considered production-ready with respect to performance or functionality. The baseline image, when coupled with hardware, can be a starting point for developers who wish to create and test software in a GENIVI context.

Synchronously with each compliance specification release (that is to say twice a year in April-May and October), the Baselines Integration Team (BIT) of GENIVI proposes at least two baselines proved to run on a Personal Computer simulation (Qemu environment) and respecting GENIVI compliance specification.

Baselines projects

These two baselines are shared in Open Source Projects. They both use built environments that are open and maintained by the Open Source community.

<table>
<thead>
<tr>
<th>GENIVI Baselines are Open Sources Projects, all the details on these can be found through following links:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Yocto Baseline</td>
</tr>
<tr>
<td>• Baserock Baseline</td>
</tr>
</tbody>
</table>

If you have built the GENIVI Demo Platform as proposed in chapter 1, you have used one of these baselines.

Automakers, their Tier1 suppliers, Software companies propose other builds that are GENIVI compliant as well, and become other proof points for GENIVI compliance specification quality.

You can learn more about GENIVI Baselines on GENIVI Baseline flyer.
4_GENIVI Components and Interfaces (API)

Rationale

GENIVI launches and manages Open Source Projects for automotive software components. Generally, GENIVI prefers to work in existing, upstream projects, but in many cases, the required automotive software is not yet available in other open source communities. GENIVI fills this gap in the automotive software ecosystem by hosting open code development projects for this automotive software.

Value

GENIVI components are standard implementations used in all GENIVI Compliant builds. The burden to define and maintain these components is shared by a community of adopting companies. As this software is used by multiple adopters, the quality of the software improves as bugs are found and fixed by those adopting companies.

Description

GENIVI leverages an Open Source Software ecosystem and integrates existing components as much as possible. GENIVI prefers to work in the "upstream first", ensuring that we do not reinvent existing components already available. The priority will be to use them and propose needed improvements to maintainers of existing Open Source Projects. Some needs for IVI are not covered upstream so GENIVI has launched and manages dedicated Open Source Projects.

GENIVI's policy is also to allow competition where it brings value and to define standards where it make sense. Some of GENIVI's Components are implementations of standard interfaces (APIs) called "abstract components" that allow for integration of alternative commercial products that provide the defined features of the component. This gives automakers and their suppliers more choice in piecing together their preferred IVI solutions.

In the table below, the links to GENIVI Open Source Projects for Components and Interfaces are presented. The blocks are consistent with the architecture block diagram shown earlier. You may navigate to these projects through the links below or use the GENIVI Open Source projects main page.

If you have any questions about GENIVI Open Source Projects, you can look at the community office page or send an email to the community manager.

<table>
<thead>
<tr>
<th>Radio &amp; Tuners</th>
<th>IVI Navigation</th>
<th>Media Manager</th>
<th>Internet functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IVI Radio</td>
<td>IVI_Positioning</td>
<td>Media Control</td>
<td>Web browser PoC</td>
</tr>
<tr>
<td></td>
<td>IVI_NavigationandMapViewer_</td>
<td>Media Indexer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IVI_PoiService</td>
<td>Media Browser</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CE Device Integration</td>
<td>Vehicle Interface</td>
<td></td>
</tr>
<tr>
<td>IVI Speech</td>
<td>Smart Device Link</td>
<td>Vehicle Web API</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Driver Workload Assessor</td>
<td></td>
</tr>
<tr>
<td>Audio Management</td>
<td>Graphic support</td>
<td></td>
<td>IPC</td>
</tr>
<tr>
<td>Audio Manager</td>
<td>IVI Layer Management</td>
<td>Common API Runtime</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wayland IVI Extension</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Persistence</td>
<td>Lifecycle</td>
<td></td>
<td>Housekeeping</td>
</tr>
<tr>
<td>Persistence Client Libr.</td>
<td>Node state Manager</td>
<td>Diagnostic Log and Trace</td>
<td></td>
</tr>
<tr>
<td>Persistence Admin. Service</td>
<td>Node Startup Controller</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Node Health Monitor</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5_Get Familiar with GENIVI Tools

Rationale

GENIVI uses a community-based development methodology which is common among open source developers. Common tooling is needed to bring consistency in the code delivered by developers from many different companies.

Value

GENIVI tools help developers be more productive in a community-based development environment. GENIVI tooling also simplifies the coding of interfaces through interface generation and translation.

Description

Some Open Source tools were created and are maintained by GENIVI.

Franca is a tool for defining and transforming software interfaces. It provides a standard way to document interfaces which is programming-language agnostic. It is widely used in GENIVI.

CommonAPI C++ is a C++ framework for interprocess and network communication. The basic objective is not to provide a new mechanism for interprocess or network communication (IPC), but to define a high-level C++ API, which can be used for different mechanisms.

YAMAICA Yet Another Model And Interface Conversion Application, is a collection of Eclipse features which implement transformations between different interface description languages, editors and code generators.

IoNAS is a proof of concept to show that from an Franca interface description, you can generate .arxml AUTOSAR RTE descriptions as well as CommonAPI. Linux-Programs can communicate with AUTOSAR-Applications.

GENIVI also uses existing Open Source and commercial tools. The complete view can be found on GENIVI Tooling main page here.
Appendix 1: Understand Open Source SW development model

Rationale

Open Source development model is very powerful but needs to be well understood and properly deployed in your company.

Value

This section will explain best practices of planning for, deploying, and managing open source software in your company.

Description

Why is it important to contribute?

An Open Source ecosystem is based upon a win-win development model.

Automakers know very well that sharing the load of development with partners who are also their competitors is a sound business practice. They are used to do it through one to one contracts on car models, platforms or engines. These contracts precisely define the roles and responsibilities of both parties and what they get from the cooperation.

What's new in Open Source software model is that the win-win situation is not “contracted” in the same way. Open source licenses define some rules (see next section) but the development is based upon the good will of the many partners involved.

The biggest temptation of the Open Source model is to take useful software but not give back bug fixes or software your company has developed. You could get the benefits but no one else benefits from your work. But this approach is short-sighted and not as beneficial as it may seem. If you take a software component from an open project (called branching) and then you keep your modifications and improvements for your own, then you will have to maintain that branched software for decades. On the other hand, if you push your proposals of improvements to the community (called “upstreaming”) and they get accepted, the code you wrote will be tested and improved by hundreds of engineers for free.

This Open Source ecosystem approach has proved its efficiency in several industries. But one should not neglect the need to change the cultural mindset when deploying Open Source in a company, plus the need to adapt some processes to this new way of development.

IVI Serial development projects are big machines involving hundreds of people and multiple companies over years. Organization, contracts, work breakdown structure are key entry data of development process that will guide behaviors of all parties along the project. These entry documents are prepared during the RFQ process when an OEM starts discussions with suppliers. People do not start from a blank sheet of paper to prepare these documents but use references that they have in their company. If we want to get the expected “upstream attitude” during development, the root document that describes how projects run needs to be updated accordingly. It is best that this root document is prepared before RFQ process. It needs to be confirmed in the organization description in the contracts all parties sign. The work breakdown structure needs to list such upstreaming activities so that the resource plan is defined accordingly.

During development, resource managers have to fight with daily difficulties; pushing bug fixes and enhancements into upstream projects cannot be their first priority. Managers will not allocate time to this important activity unless it is a standard established at the start of the program. The program management should make it clear that contributing back to upstream projects is a rock-solid standard in the company.

The key message to conclude: if you want to make your Open Source projects sustainable, make sure your company is organized to contribute.

License management

Open Source code may be free but it not free from obligations on the usage and distribution of the code. When the author of the code, who owns the copyright and any other intellectual property rights associated with the code, chooses to attach an Open Source license to his code, he has defined those obligations to anyone who uses the code.

Hundreds of Open Sources licenses exist, with a large variety of obligations to fulfill. For example, simple licenses mention the name of the author and give access to the Source Code by the final customer. Others raise more difficulties like the obligation to let the final customer to modify a product based on the code. The licenses also define the rules when some piece of code is modified : can it become commercial, should it remain Open Source code? There are also some tricky rules about code integration and some licenses may “propagate” to code into which it is integrated.

The important message is not to be frightened, but to realize that license compliance must be managed. GENIVI is not only a good example of sound license compliance management, but many of its members can help an organization plan for the open source processes that are essential to using and distributing open source software in a company’s products. GENIVI has defined its own process to deal with compliance and has worked with Tier one suppliers to understand their difficulties and fears on this topic. GENIVI has defined its own licensing policy and works with preferred licenses. The License Review Team manages the licensing and legal activities of the alliance and is chartered to ensure that the technical work of the alliance is founded on legally sound approaches that are acceptable to the membership of the alliance and to the broader open source community.

During most GENIVI All Member Meetings, GENIVI will give a primer on how to work with open source software. In a past All Members
Meeting, GENIVI had the Linux Foundation deliver a course on Open Source License Compliance:

https://training.linuxfoundation.org/linux-courses/open-source-compliance-courses/overview-of-open-source-compliance-end-to-end-process

Following this course is a great help to understand what governance and processes you need to install in your company to manage Open Source compliance and avoid troubles.

GENIVI networking in the Open Source community is also very useful. GENIVI in the past could alert members that compliance issues were raised on a product and help solve these.
Appendix 2: GENIVI Roadmap

Rationale

It is important to understand GENIVI momentum and not only understand what GENIVI is but also foresee what GENIVI will be in the next months or years.

Value

Get a detailed view on where the standardization and code production will occur in the next releases.

Description

GENIVI Releases

GENIVI uses a time-boxed approach to development. A new release appears every 6 months and coincides with All Member Meeting events. The releases have code names that are ordered alphabetically. The names have a spaceship- or astronomy-related theme.

- **Apr 2016** - Compliance Release 10.0 - Baseline Release LEVIATHAN
- **Oct 2016** - Compliance Release 11.0 - Baseline Release MIRANDA

The baselines versions when released get available in GENIVI Open Source Projects Yocto Baseline and Baserock Baseline

Future content roadmap

GENIVI offers a toolbox which allows to accelerate and simplify development of Infotainment systems through standardization and reuse of code components. This toolbox is not complete and also needs to adress a moving target. To define its roadmap GENIVI uses multiple entries:

- lessons learned from serial product developments allow to identify gaps and holes.
- member companies involved in expert groups raise new topics that they would address anyway for themselves and that they prefer to share with the community.
- board members drive the strategic trends of GENIVI scope.

These entries are shared between Expert Groups and Project Management Office, then each Expert Group defines its Roadmap in its domain. If you need more background on GENIVI organisation you can get the information there: [http://www.genivi.org/how-genivi-works](http://www.genivi.org/how-genivi-works), [http://projects.genivi.org/what](http://projects.genivi.org/what)

Detailed roadmap on each component is available in the Compliance Specification document which is part of the members benefits. If your company is not a member yet you can find member benefits at [http://genivi.org/join](http://genivi.org/join). Promote these benefits in your company and join us!

**GENIVI Platform Compliance Specification 9.0** (to be released during cw 43, 2015).
Appendix 3 : One step in Architecture

Rationale
This appendix provides a brief introduction to each area (domain) included in the scope of the reference architecture block diagram.

Value
The description will help developers to better what component is in which domain and thus avoid misunderstanding and confusion.

Description
The domain areas are defined mostly per function area.

Persistence
The Persistence Management subsystem is responsible for providing a shared solution to persistent data storage. Persistent data includes all data that is dynamically changed but needs to be stored on a head unit between restarts. It includes for example the state of user parameters when the system shuts down (what media is on, sound level and balance, historical data on destinations and phone calls,...). All types of dynamically changing but persistently stored data is expected to use the persistence subsystem. It does however not deal with software code, navigation map data and similar databases, which would typically be handled by Software Management.

Software Management
Software Management manages the storage of all program code running on an ECU including how to download and update programs, via standardized automotive diagnostic protocols, from storage devices or over a network, e.g. Software-over-the-air (SOTA).
A software management implementation updates different parts of the system and handles issues like rolling back unsuccessful updates to a known stable state. All the required steps like verifying software integrity/authenticity, system compatibility and similar issues need to be taken into account.

An Infotainment ECU frequently contains many different types of hardware, such as individual radio tuners, Bluetooth modules and other peripherals, some of which in turn need to be loaded with their own software or firmware using special protocols.

**Lifecycle**

Lifecycle deals with the startup, shutdown and management of the running of system components and integrated applications. It also keeps track of the state, and the status (health) of the system, including:

- Interfaces to/of Thermal- and Supply management.

**User Management**

User Management is the support for multiple independent users, each having their own profiles and settings, and the mechanisms for keeping track of who is active (logged in) on different parts of the system. In combination with other subsystems it also cares for protection of personal data from unwanted disclosure in its interface to Personal Information Management subsystem.

The reference architecture describes some technical approaches for creating a multi-user system and solutions for storage and separation of private data and how to use GENIVI provided components to achieve this.

**Housekeeping**

This domain groups typical software modules that monitor software behavior and handle errors.

It includes **Automotive Diagnostics Log and Trace (DLT)**, which is one of the earliest GENIVI software. It is an advanced log and trace framework that provides interfaces for applications to log messages at different levels from debug, info, to critical errors. In comparison to Linux mechanisms used in server and desktop systems like syslog / journald, DLT differs primarily by being highly optimized for embedded systems, primarily designed (not as secondary addition) for moving log messages out over a network to another node. Most of all however, DLT follows an automotive logging standard from AUTOSAR which makes it appropriate to integrate in a larger context of automotive systems and AUTOSAR tooling. Combinations with other Linux technologies are of course still possible. DLT is often used in GENIVI components for logging / tracing and is a much appreciated development tool.

**Security Infrastructure**

This subsystems groups all cryptography libraries and other support functions for security solutions, abstractions and libraries for interacting with Hardware Security Modules (HSM), the management of signatures and encryption, intrusion/anomaly detection, and a Mandatory Access Control solution implemented by a Linux Security Module (LSM).

**Diagnostics**

This domain implements automotive diagnostics features. Unified Diagnostic Services (UDS) are codified in ISO 14229-1:2013 and allow diagnostics to control functions on an in-vehicle Electronic Control Unit (ECU). Standard on-board Diagnostic Trouble Codes (DTCs) are used.

**Inter Process Communication (IPC)**

The domain includes standards for internal communication such as d-bus, any additional infrastructure to support communication (such as a message broker, if used). It also includes the Common API Runtime. The domain includes Common API Runtime. CommonAPI is an **Inter Process Communication (IPC) language binding API**, that enables applications to use different IPC middleware as backend without any changes to the application code. The current primary implementation is C++ based although a C-based variant is being developed. CommonAPI is tightly connected to Franca IDL - tools require Franca IDL described input and the communication semantics must match. To read more about Franca, see the project website.

The currently well-tested communication backend for CommonAPI C++ is D-Bus while some companies make their own backends. Future development may support kdbus as an alternative.

**Networks**

The Infotainment system often plays the role of a hub in a vehicle as it is connected to vehicle network on one side and to external resources on the other side (smart devices, cloud, ...). A large set of network technologies and protocols are envisioned (CAN, Flexray, USB, Wifi, NFC, AVB, ...).

A SOME/IP communication stack, and connection to CommonAPI is available. Whatever the type of IVI system we expect there is always some connection to a vehicle network. At the maximum the IVI system implements a user interface for lots of vehicle related functions including climate, seats, turning on and off driving features, powertrain settings and many other domains. At the very minimum, the power states (sleep, wakeup, etc.) of the ECU are typically controlled by another node in the electrical system.

The Networks subsystem is intended to group the implementations of different network technologies. Note however that Bluetooth is treated
in its own subsystem. The basic implementation of Ethernet AVB support also resides here, while it is used by Audio Management and occasionally video subsystems.

**Network Management**

Network Management includes all the infrastructure required to set up, take down, monitor and control network connections. Connection management (using connman), and helper libraries like Download Upload Messaging Manager (DUMM) are in this domain. Depending on the system, things like traffic shaping, firewalling and similar features would also be considered part of this subsystem.

**Graphics Support**

This includes graphical support libraries for rendering and compositing. Examples are the [IVI Layer Manager](#), Compositor implementations based on Wayland protocol, [IVI extension to Wayland](#), OpenGL and similar low-level graphics standards. Prominent image processing libraries could be put explicitly into this category, but just as often they may be just considered part of the "Generic libraries" section of the platform.

**Audio/Video Processing**

This subsystem includes the codecs and infrastructure for putting together a processing pipeline for audio or video. Typically this means audio and video codec implementations, or interface libraries to such implementations if they exist in specialized hardware, optimized DSP code or similar. GStreamer provides pipeline/processing support.

**Audio Management**

The Audio domain deals with management, connection and routing of audio streams taking into account unique automotive (IVI) audio requirements (priorities, mixing, foreground/background, etc.). The primary implementation is the GENIVI [AudioManager](#) framework which consists of a shared implementation of the AudioManagerDaemon plus typically a set of product specific implementations of the plug-in abstract components. In this group we typically also include some available Linux technologies for the management of audio (pulseaudio, alsal).

**Device Management**

Low level support function reacting to and distributing events for connected devices ranging from simple USB memories up to connected Bluetooth devices. An interesting challenge to implement in this subsystem is preparation of the higher level logic that may be required to react to device connections, such as handing over from one connection (wireless) to another (physical) originating from the same device. A typical implementation may require product unique logic but the platform solution should consider how general patterns can be provided to allow reuse between projects.

**Bluetooth**

Typically a full-featured Bluetooth Stack is expected in a modern IVI system to support use cases such as data exchange, networking, media streaming and hands-free telephony. Because of its importance and its many parts and features Bluetooth software components are in their own separate subsystem.

**Camera Functions**

This subsystem contains code that implements, or supports implementing, camera based functions such as rear-view (reverse gear) camera, camera assisted parking, surround-views, etc.

**Speech**

Different components, from the basic technology libraries to databases, and dialog-control, and the necessary connections to other HMI functions make up a working speech subsystem.

**HMI Support**

HMI (Human Machine Interface) includes all technologies that the user interacts with. In addition to graphical displays, this includes buttons and knobs, speech control, gestures and handwriting recognition. A subsystem is created to include implementation of button routing, recognizers for gestures/handwriting, prioritization of HMI events, etc.

There are also typically more elaborate supporting frameworks used for graphical user interfaces above the low level features provided by Graphics Support subsystem. These may include frameworks helping the implementation of GUIs (Qt, GTK, etc.). It may differ from one GENIVI system to the next which, if any, of such libraries should logically be considered part of the platform or be seen as added on top of it. This is why the reference block diagram also suggests that technologies like Qt can if desired be considered to exist outside of platform specification.

**CE Device integration**

This deals with the interaction between smartdevices and the IVI system. Some different standards include SmartDeviceLink, MirrorLink™.
CarPlay™ and Android Auto.

**Personal Information Management (PIM)**

In the context of an IVI system PIM first to keep a database of personal information about other people, such as an address book, including also phone numbers, email address, social application identities, etc. It also includes the user's own information of similar kind (web service accounts, passwords, etc.) and therefore interfaces with the user management domain.

Although some address books may be fetched or synced with Bluetooth, others may have an Internet account management elsewhere. It's assumed that many features may need access to PIM information, for example supporting a use case like navigating to the address of a contact. Designing platform components responsible for PIM enables such information to be shared and available to multiple applications instead of locked into individual ones.

In some systems PIM extends to calendar information, appointments, todo-tasks. This tends to be heavily integrated with Email, which slides over to mobile office and even social networks. Typically we group those user functions into the “Internet Functions” group, but significant interaction with PIM is likely and different products may do it slightly differently depending on required integration.

**Vehicle Interface**

This subsystem deals with any abstraction or connection to typical automotive vehicle networks, and to the signals and informations handled there. In some systems this may include connection to a peripheral CPU, which in turn is actually connected to vehicle buses. In practice designing all the software for a function together of course makes sense even if different parts are deployed on different CPUs - nonetheless it is worthwhile to remind that the GENIVI reference architecture deals mainly with the main “Multimedia” SoC, if the IVI system is a multi-CPU solution.

**Internet Functions**

This domain includes a traditional interactive Web Browser function and non-interactive renderers for HTML documents, as well interfaces to cloud based services. Very often frequently changing user features will be implemented as "apps" on top of a platform, but in any system where these are considered integrated into the platform, then features like email, social networks etc. may also be included in Internet Functions subsystem.

**Media Sources**

This platform subsystem includes the actual implementations that access media data from different sources, typically presenting their result in a uniform way to the Media Framework.

Network streams and connected devices (DLNA®, Bluetooth, Internet streaming), as well as other removable media and embedded storage are typically included, but broadcast technology such as AM/FM, SiriusXM, DAB, Television broadcasts, are grouped into another subsystem.

**Media Framework**

This is a platform support for media applications which includes the generic logic that can be shared between Media players and is extended by the sources provided by the Media Sources subsystem.

A media framework supports media applications with the ability to connect and integrate different sources of media as a combined resource.

In addition to playback logic, primary features include maintenance of music metadata, music identification services and coordination of media playback.

**Navigation & Location Based Services (LBS)**

This domain deals with traditional IVI navigation systems and the supporting technology for these such as positioning

It also includes all data services and innovative features that have location as their primary feature.

**Telephony**

This is the telephony software stack. It complements the Bluetooth communication (typically) with the actual logic for connecting and managing phone calls, call-waiting, conferences and so on. There may be multiple technologies behind such as an in-car embedded phone, or VOIP functionality and the telephony stack could (if desired) manage them similarly.

**Radio & Tuners**

This domain includes traditional broadcast technologies including receivers for radio technologies (AM/FM, DAB, HD-Radio, SiriusXM, …) as well as television broadcasts. Note that streaming and the "Internet Radio" type of sources are addressed in the Media Sources domain. The functionality of the Tuner API is verified in the IVI Radio project.

**Applications Layer**

GENIVI makes the distinction between Native Applications like Vehicle Functions, Climate, Radio, Navigation, etc and Managed Applications
that may be added by the user like Music Services, Weather, Social Networks, etc.

The exact features implemented in the Native and Managed domain respectively are decided by the system builder - GENIVI has only provided some typical examples. In fact, it's optional to use the Managed or Native or (recommended) combining both approaches.

The reason and rationale behind the two design philosophies are described in more detail in the Reference Architecture document.