Core Documentation

In the Core set of guides you will find all of the information you will need to customize and use the SDL embedded component (SDL Core) to work well with your vehicle.

First of all start with SDL Core SW Architecture document, which provides overview for the SDL technology and SDL Core product.

For Integration purpose, please follow:

- Deployment schema
- Operational aspects:
  - Configuration
  - Logging
  - Diagnostics
- Transport Manager Programmers Guide
- Preloaded Policy Table configuration
- Video Stream Setup

For Development purpose, please follow:

- Transport Manager Programmers Guide

For Testing purpose, please follow:

- Logging and diagnostics
Table of contents

• 1. Introduction
  ◦ 1.1. Purpose and Scope of the SAD
  ◦ 1.2. Definitions and Abbreviations
  ◦ 1.3. Document Roadmap

• 2. Case Background
  ◦ 2.1. System Context, Mission and Scope
  ◦ 2.2. Product Stakeholders
  ◦ 2.3. Business Goals
  ◦ 2.4. Significant Driving Requirements

• 3. Solution Overview
• 4. Views
  ◦ 4.1. Use Case View
  ◦ 4.2. Components View
  ◦ 4.3. Component Interaction View
    ▪ 4.3.1. Transport layer
    ▪ 4.3.2. Protocol layer
    ▪ 4.3.3. Business layer
  ◦ 4.4. User Interface
  ◦ 4.5. Data View
  ◦ 4.6. Process State View
  ◦ 4.7. Process View
  ◦ 4.8. Development View
    ▪ 4.8.1. Implementation Technologies
    ▪ 4.8.2. Modules and Code Base Organization
    ▪ 4.8.3. Development Environment and Standards
  ◦ 4.9. Deployment View
  ◦ 4.10. Operational View
1. Introduction

1.1. Purpose and Scope of the SAD

This document defines the high-level software architecture for the SmartDeviceLink (SDL) system. It describes the structure and the main components of the system, the project basis and dependencies. The goal of the document is to describe, in sufficient detail, the software components, their responsibilities, behavior, and interfaces. This document provides support for Luxoft, Ford, open-source developers and others to learn system design, limitations, stakeholders, and ways of extension and further development.

1.2. Definitions and Abbreviations

Abbreviations used in this document please find in the table below.
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT</td>
<td>Bluetooth</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HMI</td>
<td>Human Machine Interface</td>
</tr>
<tr>
<td>IVI</td>
<td>In-Vehicle Infotainment</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>SDE</td>
<td>Software Development Environment</td>
</tr>
<tr>
<td>SDL</td>
<td>SmartDeviceLink</td>
</tr>
<tr>
<td>SEE</td>
<td>Software Engineering Environment</td>
</tr>
<tr>
<td>TTS</td>
<td>Text To Speech</td>
</tr>
<tr>
<td>VR</td>
<td>Voice Recognition</td>
</tr>
</tbody>
</table>

Definitions used in this document are in the table below.
### 1.3. Document Roadmap

The SW architecture of system is considered from different viewpoints:

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concern</td>
<td>A functional or non-functional requirement.</td>
</tr>
<tr>
<td>Model</td>
<td>A particular diagram or description constructed following the method defined in a viewpoint. These provide the specific description of the system, which can include identifiable subsystems and elements.</td>
</tr>
<tr>
<td>Stakeholder</td>
<td>An individual, group or organization that has at least one concern relating to the system.</td>
</tr>
<tr>
<td>VIEWPOINT</td>
<td>VIEWPOINT DESCRIPTION</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Component</td>
<td>Functional type of view which describes the system’s runtime functional elements and their responsibilities.</td>
</tr>
<tr>
<td>Component Interaction</td>
<td>Functional type of view which describes interactions of the system’s functional elements. Component Interaction view uses component-level sequence or collaboration diagrams to show how specific components will interact. The purpose is to validate structural design via exploration of the software dynamics.</td>
</tr>
<tr>
<td>Use Case</td>
<td>Use Case View captures system functionality as it is seen by users. System behavior, that is what functionality it must provide, is documented in a use case model.</td>
</tr>
<tr>
<td>User Interface</td>
<td>Functional type of view which describes interfaces of the system’s functional elements.</td>
</tr>
<tr>
<td>VIEWPOINT</td>
<td>VIEWPOINT DESCRIPTION</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Data</strong></td>
<td>Describes the way that the system stores, manipulates, manages, and distributes information. The ultimate purpose of virtually any computer system is to manipulate information in some form, and this viewpoint develops a complete but high-level view of static data structure and information flow. The objective of this analysis is to answer the questions around data content, structure, ownership, quality, consistency update latency, references, volumes, aging, retention, and migration.</td>
</tr>
<tr>
<td><strong>Process State</strong></td>
<td>Concurrency type of view. Process State View is used to model standard process dynamics that are independent of the loaded components. These dynamics may, for example, be part of a component management infrastructure that loads and controls components in the process. For process dynamics, it is often useful to think in terms of a standard set of states such as initializing, operating, and shutting down</td>
</tr>
<tr>
<td><strong>Process</strong></td>
<td>Concurrency type of view. Process View describes processes and process inter-communication mechanisms independent of physical hardware deployment</td>
</tr>
<tr>
<td>VIEWPOINT</td>
<td>VIEWPOINT DESCRIPTION</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Development</td>
<td>Describes the architecture that supports the software development Process. This view addresses the specific concerns of the software developers and testers, namely code structure and dependencies, build and configuration management of deliverables, design constraints and patterns, and naming standards, etc. The importance of this view depends on the complexity of the system being built, whether it is configuring and scripting off-the-shelf software, writing a system from scratch, or something between these extremes.</td>
</tr>
<tr>
<td>Deployment</td>
<td>Describes the environment into which the system will be deployed and the dependencies that the system has on elements of it. This view captures the hardware environment that your system needs (primarily the processing nodes, network interconnections, and disk storage facilities required), the technical environment requirements for each element, and the mapping of the software elements to the runtime environment that will execute them.</td>
</tr>
<tr>
<td>Operational</td>
<td>Describes how the system will be operated, administered, and supported when it is running in its production environment. The aim is to identify system-wide strategies for addressing the operational concerns of the system’s stakeholders and to identify solutions that address these</td>
</tr>
</tbody>
</table>
## VIEWPOINT | VIEWPOINT DESCRIPTION
--- | ---
Logical | Logical view focuses on the functional needs of the system, emphasizing on the services that the system provides to the users. It is a set of conceptual models.

For more information about Viewpoints refer to Architectural Blueprints The “4+1” View Model of Software Architecture:

For detailed UML diagrams notation description please refer to:
- [https://sourcemaking.com/uml](https://sourcemaking.com/uml)

### 2. Case Background

### 2.1. System Context, Mission and Scope

SmartDeviceLink system is developed to serve as a proxy between vehicle Head Unit sub-system and an Application that runs at any of compatible Mobile Devices:

- A Mobile Device might be connected via USB, Bluetooth or Wi-Fi to the HU;
- The Application should be the SDL-enabled one.

The Mobile Device might be any of:
- Smartphone devices
- Tablet PCs

with operational system:
- iOS
- Android.
The SDL system allows Application to:
- Use vehicle HMI: VR, TTS, buttons (physical and touch-screen), vehicle display, audio system, etc.
- Retrieve Vehicle Data (seat belt position, transmission shift lever position, airbag status, etc.).

### 2.2. Product Stakeholders

Actors are stakeholders that interact with product directly.

<table>
<thead>
<tr>
<th>STAKEHOLDER NAME</th>
<th>ACTOR (YES/NO)</th>
<th>CONCERN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ford Company</td>
<td>No</td>
<td>Get the SDL system with enough quality and functionality that fulfill their goals</td>
</tr>
<tr>
<td>PM / Architect / Analyst</td>
<td>No</td>
<td>Use Customer Requirements Specification</td>
</tr>
<tr>
<td>Developers</td>
<td>Yes</td>
<td>Construct and deploy the system from specifications</td>
</tr>
<tr>
<td>Testers</td>
<td>No</td>
<td>Test the system to ensure that it is suitable for use</td>
</tr>
</tbody>
</table>

### 2.3. Business Goals

Luxoft delivered to Ford a prototype of POSIX compliant Applink Core in March, 2013.
To support FORD goal of successful acceptance of Applink (new name is SmartDeviceLink) Core by open source community of GENIVI consortium further enhancements will be required. The purpose of the project is to develop
component of SmartDeviceLink 4.x Core by adding new features required by Ford.

### 2.4. Significant Driving Requirements

The requirements are listed in the table below and ordered by descending of their significance from architectural solution point of view.

<table>
<thead>
<tr>
<th>#</th>
<th>Driving Requirement Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>System has to be POSIX-compliant to be easily ported on all POSIX standardized OSs.</td>
</tr>
<tr>
<td>2.</td>
<td>Transport for communication between Mobile Application and SDL system must be implemented and easily changed, replaced or added if required.</td>
</tr>
<tr>
<td>3.</td>
<td>APIs for communication between Mobile Application and SDL system described in appropriate documents have to be fully supported by the system.</td>
</tr>
<tr>
<td>4.</td>
<td>There has to be relatively easy way to port existing HMI Modules (such as UI, VR, TTS, etc.) to work with SDL system.</td>
</tr>
<tr>
<td>5.</td>
<td>APIs for communication between SDL system and HMI Modules have to be fully described in appropriate document and fully supported by SDL system.</td>
</tr>
</tbody>
</table>
3. Solution Overview

The picture below shows SmartDeviceLink technology overview.

SEQUENCE DIAGRAM

SOLUTION OVERVIEW

https://d83tozu1c8tt6.cloudfront.net/media/builds/master/6/01168f3eb317c052ad754898d9472f4fcc09a29b/software-architecture-document/solution-overview/assets/image14.png

4. Views

4.1. Use Case View

The following Use Case diagrams show the actors, the processes and their interactions within SDL System.
OVERVIEW USE CASE DIAGRAM

Overview

DISCONNECT USE CASE DIAGRAM

Disconnect

CONNECTION USE CASE DIAGRAM

Connection

SERVICE DATA TRANSFERRING USE CASE DIAGRAM

Service data transferring

ENCRYPTION USE CASE DIAGRAM

Encryption
DATA VERIFICATION USE CASE DIAGRAM

Data verification

RPC USE CASE DIAGRAM

RPC

MOBILE TO HMI RPC PROCESSING USE CASE DIAGRAM

Mobile to HMI RPC processing

HMI TO MOBILE RPC PROCESSING USE CASE DIAGRAM

HMI to Mobile RPC processing

RESUMPTION USE CASE DIAGRAM

Resumption
4.2. Components View

The view is represented by module and subsystem diagrams that show the system's export and import relationships. The Components View diagram and its elements description please see below.

Note: UML notation for this Components View diagram is extended: both component and it's interfaces are highlighted with the same colour.
Elements description
Utility components:

**LIFE CYCLE**

- **Responsibility:**
  - Functional components manipulation
  - creation
  - destruction
  - initialization
  - start, stop
  - binding
  - System and Utils-specifics initialization
  - Relations
  - Composes all available components

- **Interfaces**
  - Does not provide any external interfaces

- **Behavior**
  - *Life Cycle* creates all available in system components according configuration, binds components to components and starts each component internal routines.

- **Constraints**
  - N/A

**CONFIG PROFILE**

- **Responsibility**
  - Storing information about application configuration.
• Relations
  ◦ Used by Life Cycle for filling other components Settings

• Interfaces
  ◦ Provides Profile interface

• Behavior
  ◦ Config Profile parses configurable data storage and provides primitive types by section and name of configurable value.

• Constraints
  ◦ Configuration format - INI file.

UTILS

• Responsibility
  ◦ Encapsulation system low-level functionality.

• Relations
  ◦ Used by all components.

• Interfaces
  ◦ Logger macros-es and functions
  ◦ Data and Time
  ◦ Files
  ◦ Thread and Timer
  ◦ Locks and ConditionalVariable classes
  ◦ CustomString class for UTF8 string handling

• Behavior
  ◦ Utils behavior relates to system-specific API.

• Constraints
  ◦ N/A
HMI layer components:

**HMI MESSAGE HANDLER**

• **Responsibility**
  ◦ Formatting message to and from unified protocol-API-independent format used by higher-level component.
  ◦ Providing adapters for different transport types between SDL and HMI.

• **Relations**
  ◦ Application Manager
  ◦ Utils

• **Interfaces**
  ◦ **HMIMessageObserver** interface for listening HMI messages notification
  ◦ **HMIMessageSender** interface for sending Messages
  ◦ **HMIMessageAdapter** interface for abstracting to-HMI transport
  ◦ **HMIMessageHandler** interface for accumulating **HMIMessageObserver, HMIMessageSender** and **HMIMessageAdapter**

• **Behavior**
  ◦ Transferring RPC Messages between business-layer and configured transport.

• **Constraints**
  ◦ Processes messages from a single instance of HMI only.
HMI-transport need to be statically configurable with build flags.

**Business layer components:**

**APPLICATION MANAGER**

* Responsibility

- Storing and providing mobile-related information
- Mobile application state manipulation

* Relations

- Uses *Commands*
- Uses *MediaManager*
- Requires *HMIMessageObserver* and HMIMessageSender (HMI Message Handler)
- Requires *PolicyHandler* and PolicyHandlerObserver (Policy)
- Requires *ProtocolHandler* and ProtocolObserver (Protocol Handler)
- Requires *ConnectionHandler* and ConnectionHandlerObserver (Connection Handler)
- Requires *SessionObserver (Connection Handler)*
- Requires *SecurityManagerListener (Security Manager component)*

* Interfaces

- Provides *ApplicationManager* interface

* Behavior

- The component implements business logic of the SDL.

* Constraints

- N/A
**COMMANDS**

* Responsibility
  
  ◦ Mobile and HMI RPC data verification according to business-requirements
  ◦ Transferring Mobile RPC Requests to HMI subsystems (UI, VR, TTS and other available ones) and HMI to Mobile Responses and Notifications

* Relations
  
  ◦ Created by **ApplicationManager**
  ◦ Composed by **RequestController**

* Interfaces
  
  ◦ Provides **Command** interface

* Behavior
  
  ◦ Mobile Requests are spitted between responsible HMI interfaces and sent as separate HMI Requests or Notifications.
  ◦ HMI Responses and notifications are verified according to business requirements and provided to Mobile.

* Constraints
  
  ◦ FORD Mobile API Spec
  ◦ FORD HMI API Spec
  ◦ Commands happy paths are depends on correct HMI Behavior implementation.

---

**REQUEST CONTROLLER**

* Responsibility
  
  ◦ Pending requests handling
Optimization threads count for handling large quantity of pending RPCs

**Relations**
- Composes *Commands*
- Composed by *Application Manager*

**Interfaces**
- Provides *Request Controller* interface

**Behavior**
- *Request Controller* handles timeout of responses and notifications from HMI.

**Constraints**
- Configurable count of threads usage.

**APP LAUNCH**

**Responsibility**
- Launch known applications on devices.

**Relations**
- Composed by *Application Manager*
- Use *Resume Controller* interface to get HMI level of saved application.

**Interfaces**
- Provides *App Launch Controller* interface

**Behavior**
- *App Launch* launch all known applications on newly connected device.
• **Constraints**
  ◦ Not work for Android apps.
  ◦ Not work for apps connected via SDL protocol version lower than 4.

**RESUMPTION**

• **Responsibility**
  ◦ Restoring application data
  ◦ Storing application and HMI-related data between shutdown cycles

• **Relations**
  ◦ Composed by *Application Manager*

• **Interfaces**
  ◦ Provides *Resume Controller* interface

• **Behavior**
  ◦ *Resumption* backs up application and HMI-related data and restores it after SDL start-up according to business logics.

• **Constraints**
  ◦ *Configurable data storage type.*

**POLICY**

• **Responsibility**
  ◦ Enabling advanced SDL functionality
  ◦ SDL APIs protection from unauthorized application usage
  ◦ Remotely manage SDL-enabled apps, including app-specific and device-specific access to system functionality
  ◦ Maintain applications permissions on the system
• **Relations**
  ◦ Uses *ApplicationManager* interface for mobile application state manipulation

• **Interfaces**
  ◦ Provides *PolicyManager* interface for policy data manipulation
  ◦ Provides *PolicyListener* interface for policy notification subscribing

• **Behavior**
  ◦ Receives data from Application manager
  ◦ Parses data - Stores in local storage
  ◦ Provides data via Application Manager to mobile device and HMI and vice-versa

• **Constraints**
  ◦ Needs to be a switchable components: dynamically by configuration file and statically by SDL build define.

**MEDIA MANAGER**

• **Responsibility**
  ◦ Audio and Video data transferring to Media sub-system
  ◦ Encapsulation binary data transferring transport

• **Relations**
  ◦ Used by *Application Manager*

• **Interfaces**
  ◦ Provides *MediaManager* interface

• **Behavior**
  ◦ Media Manager transfers raw Audio and Video data through one of the Media-adapters.
• **Constraints**
  ◦ Configurable Media-adapter usage

**Protocol layer components:**

**PROTOCOL HANDLER**

• **Responsibility**
  ◦ Control and business data distributing to appropriate sessions and service
  ◦ Control message processing
  ◦ Multi-frames assembling and disassembling
  ◦ Malformed packets determination and filtering

• **Relations**
  ◦ Notifies *ConnectionHandler* about connection and session state change
  ◦ Uses *SecurityManager* for encryption and decryption payload data

• **Interfaces**
  ◦ Provides *ProtocolHandler* interface for data sending and protocol layer manipulation
  ◦ Provides *ProtocolObserver* notification for subscription on protocol events.

• **Behavior**
  ◦ Decodes income raw transport data and encodes outcome RPCs according to protocol specification.


• **Constraints**
  
  ◦ SmartDeviceLink Protocol specification

**CONNECTION HANDLER**

• **Responsibility**
  
  ◦ Storing devices and connection information
  ◦ Manage starting and ending of sessions
  ◦ Providing device, connection and session information for protocol and business layer
  ◦ Manipulation with devices, connections and sessions
  ◦ Negotiation and monitoring the availability of device connections (heartbeat)

• **Relations**
  
  ◦ Requires *ProtocolHandler* for sending Control messages related to session life cycle
  ◦ Requires *TransportManager* for forwarding business layer device and connection manipulations

• **Interfaces**
  
  ◦ Provides *ConnectionHandler* interface for connection manipulation
  ◦ Provides *SessionObserver* interface for session information manipulation

• **Behavior**
  
  ◦ Connection Handler works as a proxy from business-layer to transport layer and provides additional information related to protocol sessions and services.

• **Constraints**
  
  ◦ SmartDeviceLink Protocol specification
SECURITY MANAGER

• Responsibility
  ◦ Data encryption and decryption
  ◦ TLS Handshake negotiation
  ◦ TLS Library dependency encapsulation

• Relations
  • Uses SessionObserver for setting Security information to sessions
  • Uses ProtocolHandler and ProtocolObserver for handling TLS handshake data

• Interfaces
  ◦ Provides SecurityManager interface for Security component
  ◦ Provides SecurityManagerListener interface for notification handshake event
  ◦ Provides SSLContext interface for data encryption and decryption

• Behavior
  ◦ Security Manager provides methods to establish encrypted connection to mobile.

• Constraints
  ◦ Needs to be a switchable components: dynamically by configuration file and statically by SDL build define.
  ◦ SmartDeviceLink Protocol specification

Transport layer components:

TRANSPORT MANAGER

• Responsibility
  ◦ Manages low-level connections from Mobile Applications
Transport devices and connections manipulation
  ◦ Performs device discovery
  ◦ Sending and receiving mobile messages

* Relations
  ◦ Composes \textit{TransportAdapters} according to configuration

* Interfaces
  ◦ Provides \textit{TransportManager} interface for devices and connections status manipulation
  ◦ Provides \textit{TransportManagerListener} interface for transport notification subscribing

* Behavior
  ◦ Accumulative class for all available in system devices and connections.

* Constraints
  ◦ N/A

\textbf{TRANSPORT ADAPTER}

* Responsibility
  * Transport-specific API encapsulation

* Relations
  ◦ Composed by \textit{TransportManager}

* Interfaces
  ◦ Provides \textit{TransportAdapters} interface

* Behavior
  ◦ Adopts transport searching, connecting, data transferring API for one \textit{TransportAdapters interface}. 
• *Constraints*
  - For Bluetooth BlueZ transport there are only 30 connections available due to RFCOMM channels limitations.
  - Transport Manager Programming guide

### 4.3. Component Interaction View

According to layer architectural approach (see chapter 6.1), Component Interaction View could be split to Transport, Protocol and Business layer diagrams.

#### 4.3.1. Transport layer

**Behavior:**
All device notifications are transferred through the Transport Adapter, accumulated by Transport Manager and provided for the upper levels with an unique device and connection identifier.

SEQUENCE DIAGRAM

**Transport layer notification and data transferring diagram**

https://d83tozu1c8tt6.cloudfront.net/media/builds/master/6/01168f3eb317c052ad754898d9472f4fcc09a29b/software-architecture-document/component-interaction-view/assets/image16.png
4.3.2. Protocol layer

Behavior:
Protocol layer is responsible for transferring Transport and Protocol events to the Business layer.

SEQUENCE DIAGRAM

Protocol Layer - transport notifications processing diagram

https://d83tozu1c8tt6.cloudfront.net/media/builds/master/6/01168f3eb317c052ad754898d9472f4fcc09a29b/software-architecture-document/component-interaction-view/assets/image17.png
SEQUENCE DIAGRAM

Protocol Layer - data transferring diagram

https://d83tozu1c8tt6.cloudfront.net/media/builds/master/6/01168f3eb317c052ad754898d9472f4fccc09a29b/software-architecture-document/component-interaction-view/assets/image18.png
4.3.3. Business layer

**Behavior:**
Business layer is responsible for processing all income and outcome RPC data and media data streaming.
Business layer - media data transferring diagram

https://d83tozu1c8tt6.cloudfront.net/media/builds/master/6/01168f3eb317c052ad754898d9472f4fcc09a29b/software-architecture-document/component-interaction-view/assets/image19.png

SEQUENCE DIAGRAM

Business layer - RPC processing diagram

https://d83tozu1c8tt6.cloudfront.net/media/builds/master/6/01168f3eb317c052ad754898d9472f4fcc09a29b/software-architecture-document/component-interaction-view/assets/image20.png
4.4. User Interface

Not applicable, since the User Interface is not the part of development.
4.5. Data View

The Data View shows relations between separated data types and actors that perform information processing in the system. It depicts contents of saved information and also visualizes information sources, processors and destination.

The following Diagram shows relations between separated data types and actors that perform information processing in the SmartDeviceLink.

SEQUENCE DIAGRAM

DATA FLOW DIAGRAM

https://d83tozu1c8tt6.cloudfront.net/media/builds/master/6/01168f3eb317c052ad754898d9472f4fcc09a29b/software-architecture-document/data-view/assets/image21.png
Elements description
RAWMESSAGE

- **Summary:**
  - Stores raw data with connection identifier.

- **Usage:**
  - Data primitive in *Transport Manager*
  - Used by *Protocol Handler* as a transport layer income data, connection_key identifies physical connection
  - Used by *Protocol Handler* as a business layer outcome data, connection_key identifies unique session

PROTOCOLFRAME

- **Summary:**
  - Protocol layer primitive with protocol related information.

- **Usage:**
  - Used internally by *Protocol Handler* for protocol header information prepossessing

SECURITYQUERY

- **Summary:**
  - *Security Manager* primitive type.

- **Usage:**
  - Encapsulates TLS handshake and security error data
MESSAGE

• Summary:
  ◦ Application Manager RPCs primitive type with function and correlation identifiers.

• Usage:
  ◦ Internally by Protocol Handler for protocol header information prepossessing
  ◦ As abstraction for RPCs transferring by HMI Message Helper

SMARTOBJECT

• Summary:
  ◦ SmartObject acts as a union for business-layer data and could handle RPCs data as one hierarchy object.

• Usage:
  ◦ Used by Application Manager, Commands and HMI Message Helper for RPCs data filling
  ◦ RPC's data transferring between business-layer components
  ◦ Note: SmartObjects are being validated according to MOBILE_API.xml and HMI_API.xml.

MOBILE COMMAND AND HMI COMMAND

• Summary:
  ◦ RPCs objects with validation and processing data according to business requirements
• **Usage:**
  
  ◦ *Application Manager* prepares *Mobile Requests* according to *SmartObjects* from transport layer
  ◦ *Mobile Request* prepares *SmartObject* for the next *HMI Request* object and subscribes to answer event
  ◦ *Application Manager* prepares *HMI Response* according to *SmartObjects* from HMI layer
  ◦ *HMI Request* prepares *SmartObject* for the next *HMI Request* object

**JSON::VALUE**

• **Summary:**
  
  ◦ Json library abstraction

• **Usage:**
  
  ◦ Used as a primitive for JSON format in HMI transport

**DBUS MESSAGE**

• **Summary:**
  
  ◦ DBUS message system abstraction

• **Usage:**
  
  ◦ Used as a primitive for DBUS format in HMI transport

### 4.6. Process State View

The process State view shows the global SmartDeviceLink states according to system life cycle.
Elements description

INITIALIZATION

• Behaviour:
  ° SDL creates and initializes component according to configuration file.

• Relations:
  ° If all SDL subsystems successfully started, SDL starts waiting HMI and mobile connections.
  ° If failed, SmartDeviceLink is **shutting down**.
HMI CONNECTION

• Behaviour:
  ◦ SDL waits for an HMI connection.

• Relations:
  ◦ If HMI successfully connected, SDL starts processing all mobile data.
  ◦ On receiving stop signal SmartDeviceLink is shutting down.

PROCESSING DATA

• Behaviour:
  ◦ SDL handles HMI and mobile data and proceed according to business requirements.

• Relations:
  ◦ SDL starts shutdown procedure on getting stop signal from HMI or OS.

SHUTTING DOWN

• Behaviour:
  ◦ SDL stores all resumption data, unregisters all mobile applications, disconnects from HMI and denitializes all components.

• Relations:
  ◦ Finish SDL life cycle,
  ◦ Continue processing data on getting Awake command from HMI.
4.7. Process View

Not applicable, since the developed system works within one process.

4.8. Development View

4.8.1. Implementation Technologies

• C++98 language is selected as a programming language for SmartDeviceLink as a OS and CPU architecture independent.
• *CMake* tool-chain selected as a cross-platform building tools.
• *Google Test* with *Google Mock* extension is chosen as an opensource C++ test framework.

4.8.2. Modules and Code Base Organization

Development view organizes SmartDeviceLink components into logical and abstract groups called layers. The layers describe the major tasks that the components perform. The layers have different responsibilities and different providers.

*Elements description*
OS LAYER

• **Responsibility**
  ◦ Providing high-level interface for OS and hardware resource manipulation.

• **Relations:**
  ◦ Used by all other layers

• **Interfaces:**
  ◦ Provides threads, timers, synchronization, data, time, file and logging interfaces

• **Behavior:**
  ◦ Wrapping all OS-system-specific API to C++ Objects.

• **Constraints:**
  ◦ N/A

TRANSPORT LAYER

• **Responsibility**
  ◦ Encapsulates mobile and HMI transports usage

• **Relations:**
  ◦ Protocol layer

• **Interfaces:**
  ◦ TransportManager
  ◦ HMIMessageHandler
• Behavior:
  ◦ Opens connection
  ◦ Performs device discovery
  ◦ Sends / receives messages

• Constraints:
  ◦ Transport Manager Programming guide

PROTOCOL LAYER

• Responsibility:
  ◦ Encapsulates protocol manipulation.

• Relations:
  ◦ Application layer
  ◦ Transport layer

• Interfaces:
  ◦ ProtocolHandler
  ◦ ConnectionHandler
  ◦ SecurityManager

• Behavior:
  ◦ Parses and handles messages from transport layer according to Protocol
  ◦ Notify upper level about new transport and protocol layer events
• Provides Transport Layer manipulation by upper layers

• Constraints:
  ◦ SmartDeviceLink Protocol specification

APPLICATION LAYER

• Responsibility:
  ◦ Represents main business logic implementation

• Relations:
  ◦ Protocol Layer

• Interfaces:
  ◦ ApplicationManager
  ◦ MediaManager
  ◦ Command
  ◦ RequestController
  ◦ App Launch
  ◦ Resumption
  ◦ Policy

• Behavior:
  ◦ Main business logic functionality.

• Constraints:
  ◦ FORD Mobile API Spec
  ◦ FORD HMI API Spec

4.8.3. Development Environment and Standards

• Development and testing environment for Ubuntu 14.04 LTS x32/x64
  ◦ Debug Environment: Ubuntu 14.04 LTS x32/x64, Qt 5.3
  ◦ Compiler: GCC 4.9.3 (OS Ubuntu), Lua 5.2
4.9. Deployment View

The deployment view takes into account the system's requirements such as system availability, reliability (fault tolerance), performance (throughput), and scalability. This view maps the various elements identified in the logical, process, and development views—networks, processes, tasks, and objects—onto the processing nodes.

The deployment diagram is used for modeling the static deployment view of a system.

The figure below depicts the deployment diagram for SDL system.

https://d83tozu1c8tt6.cloudfront.net/media/builds/master/6/01168f3eb317c052ad754898d9472f4fcc09a29b/software-architecture-document/deployment-view/assets/image24.png
Elements description

MOBILE DEVICE

* Short Description:
  - The SDL application model permits multiple applications to be concurrently active and connected to the HU.
  - A few of those applications may interact with the user at a time using the HMI (depending on HMI).
  - SDL uses the concept of HMI Levels to describe the current state of the application with regards to the level at which the head unit can communicate with it (and vice versa).

* Relations:
  - Receives policies updates from **Cloud Server**
  - Sends statistics to **Cloud Server**.

* Requirements:
  - Android OS or iOS.
HEAD UNIT

• **Short Description:**
  HU HMI allows the user/driver to interact with the vehicle.
  - This interface includes:
    - A set of presets
    - Media buttons (seek forward/backward, tune up/down, and play/pause)
    - Menu items
    - Graphic user interface
    - Voice commands, etc.

• The HU HMI Handler interfaces with SDL Core to support the API functionality.

• **Relations:**
  - Communicates with applications on **Mobile Device**

• **Requirements:**
  - N/A

CLOUDSERVER

• **Short Description:**
  A Server that provides information about:
  - Which applications are allowed to run in vehicle
  - What interfaces application is allowed to use.
  In addition, server provides:
  - System configuration, including the time of the next file update
  - Some important server information to the back end user

• **Relations:**
  - Sends policies updates to **Mobile Device**.
  - Receives statistics from **Mobile Device**.
• Requirements:
  ◦ N/A

### 4.10. Operational View

This view describes how the architecture provides the ability for operation/support teams to monitor and manage the system. To make system more flexible and to support different platforms, SW provides a configuration and logging components, which are able to change system behavior according to settings defined in smartDeviceLink.ini file and to diagnostic.

### SDL CONFIGURATION

*Config Profile* component specifies the desirable system behavior on different platforms and provides settings parameters for each functional component or functionality:

- Mobile and HMI transports connection
- Protocol, Connection, Security
- Policy, Resumption
- File system restrictions
- Global properties like HelpPrompt, TimeoutPrompt, HelpTitle, HelpCommand
- Default Timeout for mobile application commands
- Desirable location of the system data (log files, persistence data, temporary data)

For further information with a list of all available parameters please refer to chapter "15.1 SDL’s configuration file structure" of HMI Guideline or smartDeviceLink .ini file.
LOGGING CONFIGURATION

SDL logging system (with a log4cxx library for posix build) provides powerful flexibility and allows to configure SDL for development, integrator and user purposes by changing log4cxx property file. Each SDL component can be configured with own:

• Logging level output
  ◦ Example: for user needs using Warning+ level is preferable for all OSm Transport and Protocol layers components.

• Output source appender
  ◦ SDL (with Log4cxx) can log to the console, files, remote socket servers, NT Event Loggers, remote UNIX Syslog daemons and others.

• own output log pattern

For further information about configuration please refer:

• log4cxx HowTo
• Configuring loggers

DIAGNOSTICS

SmartDeviceLink system provides diagnostics messages log file with following types of messages:

• Fatal message indicates abnormal problem related to external subsystems contract violation or SDL implementation issues. It indicates some critical issue and all SDL behaviors is undefined after this message.
• **ERROR** message shows, that the problem occurred and SDL has not accomplished some internal or API activities. Error is successfully handled by SDL, but notifies about some business logic's flow breakdown.
• **WARNING** message warns against uncommon or rare flow. This message indicates handling some expected by SDL issue according to specified requirements.
• **INFO** informs SDL user, integrators or support engineer about the component high-level activity success.
• **DEBUG** and **TRACE** messages contain debug information for software engineer diagnostics and deep issues analysis.

For further information about logging levels usage please refer related article.

5. View-to-View Relations

Each of the views specified in Section 3 provides a different perspective and design handle on a system, and each is valid and useful in its own right. Although the views give different system perspectives, they are not independent. Elements of one view will be related to elements of other views, and we need to reason about these relations.

5.1. Component-to-Layer

The following table is a mapping between the elements in the Component view and the Development view. The relationship shown is *is-implemented-by*, i.e. the layers from the Development view shown at the top of the table are implemented by any selected elements from the Component view, denoted by an "X" in the corresponding cell.
<table>
<thead>
<tr>
<th>COMPONENT / LAYER</th>
<th>APPLICATION LAYER</th>
<th>PROTOCOL LAYER</th>
<th>TRANSPORT LAYER</th>
<th>OS LAYER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Life Cycle</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Config Profile</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application Manager</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Commands</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Request Controller</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>App Launch</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resumption</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Policy</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Media Manager</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Protocol Handler</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
### 5.2. Data-to-Layer View

The following table is a mapping between the elements in the Data view and the Development view. The relationship shown is *is-implemented-by*, i.e. the layers from the Development view shown at the top of the table are implemented by any selected elements from the Component view, denoted by an "X" in the corresponding cell.

<table>
<thead>
<tr>
<th>Component / Layer</th>
<th>Application Layer</th>
<th>Protocol Layer</th>
<th>Transport Layer</th>
<th>OS Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection Handler</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security Manager</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HMI Message Handler</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Transport Manager</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transport Adapter</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Utils</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>DATA / LAYER</td>
<td>APPLICATION LAYER</td>
<td>PROTOCOL LAYER</td>
<td>TRANSPORT LAYER</td>
<td>OS LAYER</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>----------</td>
</tr>
<tr>
<td>Message</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SmartObject</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mobile Comm.</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mobile Comm.</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Protocoll</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security Query</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RawMessage</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JSON::Value</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DBUS message</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
6. Solution Background

6.1. Architecture Design Approach

During the architecture designing the following aspects and rules were primary considered:

1. **Three-layer architectural approach**: Transport (low), Protocol (middle), Application (high) layer.
   
   1. Each layer component uses only own or low layer interfaces
   2. **Observer** design pattern is required for providing information for upper layer components.

2. **Weak components and classes coupling** for providing SmartDeviceLink Extensibility.
   
   1. Providing each component and class functionality with an interface.
   2. **Facade** design pattern is used for Mobile and HMI transports functionality within one interface.
   3. **Observer** interface for providing information for upper layer components.
   4. Binding different layers components is in LifeCycle component responsibility.

3. **System scalability** for adding new interrogation platform transport.
   
   1. **Adapter** design pattern is used to provide permanent interface to transport layer.
   2. **Abstract Factory** design pattern is used to create related objects without specifying their concrete classes directly.
   3. **Command** design pattern is used to treat requests as an object that provides possibility to add new request without existing code modification.
4. **OS and hardware abstraction** for simplifying portability to non-POSIX-compliant OS.

   1. **Adapter** pattern is used for preparing the cross-platform interface for thread, timer, synchronization, file and data access functionality.
   2. For HMI and Mobile transports used **adapter** pattern with a unified interface, which could be reused for new platform- and OS-specific transport API adoption.
   3. OS-related and 3rd-party libraries APIs are segregated in Utils component, which available for all SDL layers.
   4. Utils classes are implemented with **Bridge** design pattern (PIMPL idiom) for avoiding platform and 3rd-party libraries dependencies.

5. **Asynchronous data and notification handling** for meeting real-time restrictions on transport layer and providing vertical scalability.

   1. Different data-types processing preferable in separate threads.
   2. For internal data processing components preferable to use `Utils::threads::MessageLoopThread` for the same data objects processing
      - Asynchronous call result has to be provided with Observers interfaces to callee
   3. For transport API adapters preferable the own `Utils::threads::Thread` implementation for meeting realtime restrictions.
   4. SmartDeviceLink logging implemented with `Utils::threads::MessageLoopThread` for avoiding console, file and other appends delay affect on functionality.
   5. For pending large number processing RPCs selected **Controller** design pattern with a limited and configurable count of processing threads.

6. **Resource Acquisition Is Initialization** (RAII, or Scope-based Resource Management) for memory, mutex, files and other resources management.

   1. `utils::SharedPtr` template class is implemented for memory deallocation.
   2. `utils::AutoLock` is implemented for mutex acquiring and releasing.
   3. `utils::ScopeGuard` is implemented for external memory and resource deinitialization.
7. **Strict heap memory usage** for avoid memory leaks and memory corruption.

   1. SmartDeviceLink objects aggregation is preferable by reference link storing instead of raw pointer,
      - Note: in case external class life-time guaranty.

   2. System objects composition is preferable by value or by smart pointer:
      1. In case of exclusive object handling could `std::auto_ptr` is preferable
      2. For shared object handling `utils::SharedPtr` is preferable

### 6.2. Requirements Coverage

There are indirect requirements which may impact on Architectural decisions, such as limitation of usage of RAM, ROM, requirements to support specific SDL Core to HMI transport layers. All the requirements of this kind were taken into account while creating Architecture Design.

- FORD Mobile API Spec
- FORD HMI API Spec
- SmartDeviceLink Protocol specification
- HMI Integration Guidelines
- SDL-Core Requirements

- Note: This requirements are handled Luxoft internally and not delivered to open-source.

### 6.3. Prototyping Results

Architecture prototyping was done to validate architecture on early stages. An evolitional prototyping technique was used. Thus all prototype components
were used with non-significant changes and additional features for further development.

6.4. Open Questions and Known Issues

List of opened questions and issues is available in sdl_core github repository:
- https://github.com/smartdevicelink/sdl_core/issues

List of Luxoft to Ford opened question is internally available in Luxoft Jira:
- https://adc.luxoft.com/jira/issues/?jql=project=APPLINK AND issuuetype=Question AND resolution=Unresolved AND labels=to_discuss ORDER BY key DESC

List of Luxoft internal questions is available in Luxoft Jira:
- https://adc.luxoft.com/jira/issues/?jql=project=APPLINK AND issuuetype=Question AND resolution=Unresolved AND labels!=to_discuss ORDER BY key DESC

6.5. Results Analysis

Not applicable, since no quantitative or qualitative analysis was performed.

7. References

2. Cmake documentation - https://cmake.org/documentation/
8. List of Figures

OVERVIEW USE CASE DIAGRAM

DISCONNECT USE CASE DIAGRAM

CONNECTION USE CASE DIAGRAM

SERVICE DATA TRANSFERRING USE CASE DIAGRAM

ENCRYPTION USE CASE DIAGRAM

DATA VERIFICATION USE CASE DIAGRAM

RPC USE CASE DIAGRAM

MOBILE TO HMI RPC PROCESSING USE CASE DIAGRAM

HMI TO MOBILE RPC PROCESSING USE CASE DIAGRAM
None
10. History

10.1. CHANGE HISTORY
<table>
<thead>
<tr>
<th>VERSION</th>
<th>DATA</th>
<th>STATUS</th>
<th>AUTHOR/EDITOR</th>
<th>CHANGE DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>04/23/2013</td>
<td>Initial Version</td>
<td>Alexandr Kandul</td>
<td>Template creation Chapter 4 'Views' creation</td>
</tr>
<tr>
<td>1.1</td>
<td>04/30/2013</td>
<td>Draft</td>
<td>Klimenko Dmitriy</td>
<td>Chapter 5 creation</td>
</tr>
<tr>
<td>1.2</td>
<td>05/10/2013</td>
<td>Draft</td>
<td>Polina Vyshnevskak a</td>
<td>Description for the Views creation</td>
</tr>
<tr>
<td>1.3</td>
<td>05/14/2013</td>
<td>Draft</td>
<td>Alexandr Kandul</td>
<td>Chapters 1,2,3 creation</td>
</tr>
<tr>
<td>1.4</td>
<td>05/27/2013</td>
<td>Draft</td>
<td>Alexandr Kandul</td>
<td>Views diagrams and description update</td>
</tr>
<tr>
<td>1.5</td>
<td>05/28/2013</td>
<td>Draft</td>
<td>Anastasiya Britanova</td>
<td>Chapters 7,8 creation formatting grammar corrections</td>
</tr>
<tr>
<td>2.0</td>
<td>05/30/2013</td>
<td>Draft</td>
<td>Anastasiya Britanova</td>
<td>Ready for customer review</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Status</td>
<td>Author/Editor</td>
<td>Change Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>----------</td>
<td>-----------------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>2.1</td>
<td>09/30/2013</td>
<td>Draft</td>
<td>Polina Vyshnevskaya</td>
<td>Changes in accordance to implementation</td>
</tr>
<tr>
<td>2.2</td>
<td>10/31/2013</td>
<td>Reviewed</td>
<td>Polina Vyshnevskaya, Pavel Savyelyev</td>
<td>Reviewed with customer</td>
</tr>
<tr>
<td>2.3</td>
<td>05/23/2014</td>
<td>Draft</td>
<td>Andrey Oleynik</td>
<td>Policy component updated</td>
</tr>
<tr>
<td>2.4</td>
<td>01/30/2015</td>
<td>Draft</td>
<td>Aleksandr Kutsan</td>
<td>Resumtion component updated</td>
</tr>
<tr>
<td>2.5</td>
<td>02/16/2015</td>
<td>Draft</td>
<td>Alexander Galiuzov</td>
<td>Update policy and SDL components</td>
</tr>
<tr>
<td>2.6</td>
<td>03/24/2015</td>
<td>Draft</td>
<td>Andrey Oleynik</td>
<td>Policy component updated</td>
</tr>
<tr>
<td>2.7</td>
<td>08/25/2015</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Ported to Luxoft Confluence</td>
</tr>
<tr>
<td>VERSION</td>
<td>DATA</td>
<td>STATUS</td>
<td>AUTHOR/EDITOR</td>
<td>CHANGE DESCRIPTION</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>---------</td>
<td>---------------------</td>
<td>------------------------------------------------------------</td>
</tr>
<tr>
<td>2.8</td>
<td>03/31/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Updated Component View diagram and components</td>
</tr>
<tr>
<td>2.9</td>
<td>04/19/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Updated Data View diagram</td>
</tr>
<tr>
<td>2.10</td>
<td>04/28/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Move Use-cases to System scope</td>
</tr>
<tr>
<td>2.11</td>
<td>05/17/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Update Development View</td>
</tr>
<tr>
<td>2.12</td>
<td>05/19/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Update Deployment View</td>
</tr>
<tr>
<td>VERSION</td>
<td>DATA</td>
<td>STATUS</td>
<td>AUTHOR/EDITOR</td>
<td>CHANGE DESCRIPTION</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>--------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>2.13</td>
<td>06/06/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Deleted Logical VIEW Update Solution Overview, View-to-View Relations, Solution Background Update references list and Operation view with diagnostics and configuration information</td>
</tr>
</tbody>
</table>
Add Windows platform from SOW
Figure Resumption updated
Add Config profile constraints
Add OS layers for View-to-view tables
Update viewpoint description
Rephrase Significant Driving Requirement 8
Process State View update with Stop transition
Add Component View highlighting notes
Add links to Viewpoints and UML notation
Updated referencing
Spelling fixes
<table>
<thead>
<tr>
<th>VERSION</th>
<th>DATA</th>
<th>STATUS</th>
<th>AUTHOR/EDITOR</th>
<th>CHANGEDescription</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.16</td>
<td>06/30/2016</td>
<td>Reviewed</td>
<td>Elisey Zamakhov</td>
<td>Minor review changes</td>
</tr>
<tr>
<td>3.0</td>
<td>06/30/2016</td>
<td>Approved</td>
<td>Nataly Snitsar</td>
<td>Approve by management</td>
</tr>
<tr>
<td>VERSION</td>
<td>DATA</td>
<td>STATUS</td>
<td>AUTHOR/EDITOR</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>--------</td>
<td>-------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>3.1</td>
<td>08/01/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Porting to Ford developers portal Update Requirements Coverage chapter Add GitHub Open Questions and Known Issues Update Components View description with Constraints Update Layers Constraints Removed assumptions from 2.4. Significant Driving Update Requirements</td>
</tr>
<tr>
<td>3.2</td>
<td>08/10/2016</td>
<td>Draft</td>
<td>Sergey Levchenko</td>
<td>Add AppLaunch Component</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Status</td>
<td>Author/Editor</td>
<td>Change Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>--------</td>
<td>-----------------</td>
<td>---------------------------------------------------------</td>
</tr>
<tr>
<td>3.3</td>
<td>08/11/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Use cases moved to a separate chapter Add applications data storing in Process State View</td>
</tr>
<tr>
<td>3.4</td>
<td>08/16/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Update Diagnostics chapter with log4cxx features usage</td>
</tr>
<tr>
<td>3.5</td>
<td>09/09/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Change Coding style</td>
</tr>
<tr>
<td>3.6</td>
<td>09/21/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Update BT BlueZ limitation</td>
</tr>
</tbody>
</table>
There are several different types of configurations for SDL that you'll have to understand in order for SDL to work properly and with the features you want on your embedded platform.

**cmake**

You'll use the cmake configuration to set up SDL before you compile, and enable or disable features like logging. The cmake file is located at `sdl_core/CMakeLists.txt`.

**smartDeviceLink.ini**

The ini file located at `build/src/appMain/smartDeviceLink.ini` after you compile and install SDL is your main configuration file for runtime configurations.
sdl_preloaded_pt.json

The policy table located in `build/src/appMain/sdl_preloaded_pt.json` after you compile and install SDL is the default policy table which provides the permissions and default configurations for SDL on its first run before it receives an update from a policy server.

NOTE

If you don't have a policy server and want to experiment with changes in the policy table, you can either edit the policy database directly with sqlite3 or edit the `sdl_preloaded_pt.json`, remove the `build/src/appMain/storage` folder, and restart SDL to load the new configuration.

The preloaded policy table located at `src/appMain` can be configured before your first run of SDL to set permissions levels and urls.

NOTE

To configure SDL using the preloaded policy table after your first run, remove the `storage/` folder from `build/src/appMain`

Let's take a look at the values that can be configured.
Module Config

The module config section contains some global defaults that can be set for SDL

**Exchange After X Ignition Cycles**

An "Exchange" is when SDL sends a request to a connected application to retrieve a new policy table from the server. This value is the number of ignition cycles before SDL initiates an exchange.

**Exchange After X Kilometers**

The distance traveled in the vehicle before SDL initiates an exchange

**Exchange After X Days**

The number of days that has passed before SDL initiated an exchange

**Timeout After X Seconds**

The amount of time SDL will wait for an exchange to complete before timing out and retrying

**Seconds Between Retries**

A list of times in seconds to wait after a failed policy table exchange before trying again. The number of items in this list determines the number of policy table retries.
Endpoints

This section is a list of URLs that is used throughout SDL.

**0x07**

A list of URLs that can be used for policy table exchanges.

**0x04**

A list of URLs that can be used to retrieve software updates.

**QUERY_APPS_URL**

A list of URLs that can be used to receive valid apps for querying on iOS devices.

**LOCK_SCREEN_ICON_URL**

A list of URLs that host an image which can be displayed by the application on the driver's device during lockout. This URL is sent in a request after each application is registered. The application proxy downloads the image and sends a notification to the application with the image to be displayed during lockout.
Functional Groupings

The functional groupings are the different named groups of rpc permissions that an application can have. There can be any number of functional groups. The functional groups are used in the next section to define behavior for different applications.

App Policies

The app policies are permissions that each application has on the system. This is where you would change the default permissions for an application, or add policies for a specific application.

General Description

As a developer implementing SDL Core into a vehicle, you may need to create new transport adapters to support custom transport implementations and libraries for Bluetooth, USB, etc. connectivity. This guide will walk you through how to create those implementations.

Terms and Abbreviations

<table>
<thead>
<tr>
<th>ID</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDL</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td><strong>SmartDeviceLink</strong></td>
<td></td>
</tr>
<tr>
<td><strong>TA</strong></td>
<td><strong>Transport Adapter</strong></td>
</tr>
<tr>
<td><strong>TCP</strong></td>
<td><strong>Transmission Control Protocol</strong></td>
</tr>
<tr>
<td><strong>TM</strong></td>
<td><strong>Transport Manager</strong></td>
</tr>
<tr>
<td><strong>UID</strong></td>
<td></td>
</tr>
</tbody>
</table>
NOTE

"Connection Establishing" with the application or device:

- Means that the transport layer creates a physical connection for sending and receiving "handshake" data
- Does not mean that this application or device shall be marked as "connected" for the user
- May be a one time connection with which data related to the device is cleared after disconnection
- May be a persistent connection with which information about the device is stored in a powered off state

"Connection Closing":

- Means that connection is not terminated until final 'goodbye' data transmission has completed
- Physical disconnection happens when there is no more data left for transmission

Transport Level Structure

Figure 1 demonstrates the structure of the Transport Level
NOTE

• The transport manager has no limitations on the number of transports
• Each transport has no limitation on the number of devices
• Any device may be connected through any number of transports. In this case each connection established between the application on a device and the transport manager has a unique identifier

SEQUENCE DIAGRAM

Figure 1

https://d83tozu1c8tt6.cloudfront.net/media/builds/master/6/01168f3eb317c052ad754898d9472f4fcc09a29b/transport-manager-programming/assets/tastructure.png

Transport Manager Structure

Figure 2 represents the structure of the Transport Manager
**Transport Adapter Structure**

Figure 3 represents the structure of a Transport Adapter.
Operation Examples

SEQUENCE DIAGRAM

New Device Search

https://d83tozu1c8tt6.cloudfront.net/media/builds/master/6/01168f3eb317c052ad754898d9472f4fccc09a29b/transport-manager-programming/assets/newDeviceSearch.png
SEQUENCE DIAGRAM

New Device Connection

https://d83tozu1c8tt6.cloudfront.net/media/builds/master/6/01168f3eb317c052ad754898d9472f4f09a29b/transport-manager-programming/assets/newDeviceConnection.png
SEQUENCE DIAGRAM

Connection Close Command

https://d83tozu1c8tt6.cloudfront.net/media/builds/master/6/01168f3eb317c052ad754898d9472f4fcc09a29b/transport-manager-programming/assets/connectionCloseCommand.png
Close connection request

Start close connection process

Send accumulated data

New data for connection being closed

Reject incoming data

Sending data is finished

Disconnect command

Disconnection is done

Disconnection is done
Transport Manager Usage

General Processing Description

**TM Initialization**

- Every time SDL starts working it calls the creation and initialization of the TM.
- The TM uses a singleton pattern and the instance of the TM is created and initialized while being retrieved. The pointer to the TM is provided to the developer.
- If TM is initialized once it must not be initialized again.
- The developer may initialize the customized TM by calling the appropriate `init()` function:
  - If the customized TM is based on default implementation, `init()` must NOT be called twice for one and the same TM.
  - If the customized TM is created from scratch, it is up to developer to choose the initialization mechanism.

- During the initialization process TM creates two threads:
  - For processing the message queue of commands from the upper level.
  - For processing the events coming from devices.

- Also TM creates and initializes all available Transport Adapters (the default and/or customized ones).
- If appropriately configured TM may load the information about previous state and perform necessary actions (e.g. reconnect the last connected application).
- TM becomes initialized:
  - When its internal threads are created and are ready for working (i.e. the `init()` function is serial and if it returns the control, the
initialization has completed successfully and TM is ready for working).

◦ And it does not watch whether the Transport Adapter(s) has initialized by this time:
  ◦ If any of TAs failed to initialize, this is informed into the log file. TM answers with erroneous messages to the requests related to such TA.
  ◦ If there are NO TAs (either not initialized or not defined), TM answers with the error messages to all of the requests.

**TM Structure**

- **TM Core:** it is a TransportManagerImpl class. It contains the processing mechanisms (e.g., for procedures of sending/receiving the data, for connecting/disconnecting procedures, etc.). It does not have the embedded default Adapters and Listeners. And it would not work apart of the Wrapper.
- **TM Wrapper:** it is implemented in TransportManagerDefault class. It adds the default Transport Adapter(s) and Listener to the Core completing the fully-featured functionality.

**TA Initialization**

- The default Adapters are created and initialized by the default TM Wrapper.
- The customized Adapters should be initialized by the developer himself:
  ◦ TM is initialized first
  ◦ Then the init() function for the custom Adapter is called
  ◦ Then the initialized custom TA and the Listener are added to the initialized TM.
- Transport adapter in its turn creates and initializes all available workers.
  ◦ ‘Search Device’ worker creates a separate thread and waits for the ‘Start’ command.
  ◦ ‘Client Listener’ creates a separate thread and waits for connections from devices.
‘Server Connection’ worker executes all actions on caller’s thread (does not create a thread).

**Getting Started**

• When the initialization is complete, TM starts waiting for:
  ◦ The user`s command
  ◦ The device connection.
  ◦ Resumption from “last state” singleton (if such resumption is specified via profile)

• When one of the above happens, TA:
  ◦ Creates a separate thread for each connection with device’ application(s)
  ◦ Notifies TM on connection created.

• When the connection is established, the Upper Level:
  ◦ May start a handshake routine with the application and then notify the user about the application is connected.
  ◦ May close the connection by sending an appropriate command to TM if the application or device is unwanted.

**Errors in TM**

• ‘Immediate’ error:
  ◦ When TM is not able to execute any command it will immediately return the appropriate error code. For example: when connect (app_id) is called and TM is not initialized yet this type of error occurs.

• ‘Postponed’ error:
  ◦ TM is able to execute a command and there is some error occurred in downstream.
  ◦ The component, where the error has occurred, sends the appropriate information to TM.
Then TM provides a notification to the Upper Level.

**Messages in TM**

- TM is ready to send and receive data after the connection is opened
- Sending messages:
  - Each message destined for the device is posted into the message queue.
  - The message is removed from the queue after it is successfully sent to the device.
  - The message(s) is returned to the caller if for some reason TM is not able to send it (e.g., unexpected device disconnection),
- Receiving messages:
  - TM redirects messages from the Transport Adapter to the Upper Level via notification mechanism.

**Connection Identifiers**

- TM uses the pair “Device ID” and “Application ID” for accessing the application on a device and internally for connection establishing.
- Device ID:
  - Stands for global unique identifier based on MAC address for network adapters and MAC address like for BT.
  - In the default implementation Device ID is logically split into two parts:
    - Internal device ID – that is a MAC address string
    - External device ID – that is an integer value.
  - When the new MAC address is found, the integer value is assigned to it correspondingly (starting from 1 and incremented with every new assignment).
  - For persistent connection
    - When the device is marked as “known”, the correspondence of internal and external IDs is stored (even after power off) until the user explicitly asks to “forget about device”.
When the user requests to ‘forget’ the known device, the MAC address may be assigned with the new integer value on being connected for the next time.

For one-time connection this correspondence is not saved in long term storage. And if connected later the same MAC address may get the new integer value in correspondence.

It is not defined what happens if two devices with the same MAC address would connect.

• Application ID: is the application unique identifier. It is the incremental integer value assigned and used internally in TM.

• Connection ID:

  • Connection is represented with a unique pair of “Device ID” and “Application ID”.
  • Connection ID is a system wide unique identifier (incremented integer value) assigned to each new connection.
  • This ID is used for sending/receiving data and for closing the connection.
  • There are no certain rules to define how exactly this ID is assigned.

**Connection Closing**

• When TM receives ‘close connection’ or ‘disconnect’ commands it tries to finish sending accumulated messages and then closes the connection.
• If connection is lost TM will drain all accumulated messages and confirm connection closure.

**SDL Shutting Down**

• When SDL is going to shutdown, TM clears all objects which it has created (e.g. default transport adapters).
• Objects created by developer (e.g. developer’s transport adapter or listener) are not removed by TM. The developer must take care of destructing his customized objects by himself.
Default TM Instance

For creating the instance of the TM with the default configuration it is necessary to use the following code

```cpp
#include "transport_manager.h"
#include "transport_manager_default.h"
{
    TransportManager *transport_manager = TransportManagerDefault::instance();
}
```

Initialization of TM may take some time due to thread creation. After the initialization is complete, the Transport Manager is ready to be used. Any commands sent before the transport manager is ready will be rejected with error code `NOT_INITIALIZED` in the return value. The TM uses the singelton pattern, thus only one instance of a TM can exist at any one time. This rule can be changed in a custom TM implementation if necessary.

Adding Custom Listeners to the TM

1. A listener allows monitoring events that take place in the TM
2. The number of Listeners is not limited and can be zero
3. The listeners are provided to the TM and are used by any module that needs to receive notifications from the TM
4. The list of Listeners is stored by the TM
5. The TM does not create any Listeners by default
6. A custom Listener should be related with the implementation of the module that uses the listener and should implement the Transport Manager Listener Interface
Adding Custom Transport Adapters to the Default TM

#include "transport_manager.h"
#include "transport_manager_default.h"

class MyTransportManagerListenerImpl: public TransportManagerListener
{
    // Implement interface
}
{
    TransportManager *tm_impl = TransportManagerDefault::instance();
    TransportManagerListener *my_tm_listener = new MyTransportManagerListenerImpl();
    tm_impl->addEventListener(my_tm_listener)
}

NOTE

About Transport Adapters in general:

• TM may contain zero to N Transport Adapters (a TM with 0 TAs does not do anything)
• Each adapter corresponds to a specific type of transport (Bluetooth, USB, TCP, etc.)
• TA implements transport specific search, connect, disconnect, and data transfer routines
NOTE

Several instances of TA:

- Several TAs of the same type can exist in the TM
- The results of using several instances of the default TAs is not defined
- It is the developer's responsibility to create a custom Adapter that operates well under these conditions

NOTE

Custom TA

The simplest way to add a custom TA is to derive it from the existing implementation of the TCP or BT adapter and add it to the TM
# Initialization

By default, the TA is initialized during TM initialization. If the TA needs to be added to a TM which has already been initialized, the TA should be initialized before it is added to the TM.

## NOTE

The TM has a TCP adapter by default. Adding the new instance of the same Adapter may result in unexpected behavior. The developer must change the standard behavior of the TCP adapter to eliminate any potential problems.
Custom Transport Adapter Implementations

If the default implementation of TAs is not suitable, you can create a transport adapter from scratch.

- Create a custom class that implements the transport adapter interface. The custom TA should use the defined interface TransportAdapterListener for notifying the Transport Manager.
- The instance of TA listener will be set by the TM automatically when a custom TA is added. If the custom TA does not store its listener then the adapter will not be able to notify the TM about events such as OnDataReceiveDone or OnConnectDone.

```cpp
#include "transport_adapter.h"

class MyTransportAdapterImpl : public TransportAdapter
{
    // Implement all interface functions here
    // use TransportAdapterListener interface to notify TM about events
}

#include "transport_manager.h"
#include "my_transport_adapter_impl.h"

{ 
    TransportAdapter *my_transport_adapter = static_cast<MyTransportAdapterImpl *>(new MyTransportAdapterImpl);
    my_transport_adapter->init();
    transportManager *tm_impl = TransportManagerDefault::instance();
    tm_impl->addTransportAdapter(my_transport_adapter);
}
```
All internal logic is up to the developer. The developer is responsible for implementing:

- Communication with the device
- Notification of state changes
- Error handling

Adding a New Listener to the transport Adapter

1. A listener allows monitoring the events that take place in the TA
2. The number of TA listeners is not limited
3. The list of Listeners that are called on any event that occurs in a TA is stored in the TA
4. A custom TA listener:
   - The developer can add the Listener to the TA through the customization procedure only. The developer needs to create his own TA on the base of the default one, and then add the Listener to it.
   - should implement the Transport Manager Listener Interface

NOTE

Working with the TA Listener by-passing the TM is dangerous and may lead to asynchronous behavior that is undefined. The custom Listener should be added only together with the new custom Adapter and/or custom TM

Set up a Listener as follows
Creating a TM with custom TAs only

If for some reason the default adapters are not a good fit they can be completely replaced with developer defined adapters. To do this the developer

1. Must implement the TA logic
2. Provide the new TA to the TM
Transport Manager Customization

1. Basic Information

- TM is responsible for all complex logic and decisions, while TA is a primitive entity that operates only with transport specifics
- Communication interface between TM and TA
- TM sends a command to TA
- If TA is unable to execute this command it returns the proper error code
- Otherwise, TA starts executing the command. The TA notifies the TM by using the appropriate callback function

```cpp
#include "transport_manager.h"
#include "transport_manager_impl.h"

class MyTransportManager : public TransportManagerImpl {
    virtual int init();
    virtual ~MyTransportManager();

    MyTransportAdapter *my_adapter_;
    explicit MyTransportManager(const TransportManagerAttr &config) : TransportManagerImpl(config), my_adapter_ (nullptr) {
    }

public:
    static MyTransportManager *instance();
};

// note: the implementation of all methods above are not defined here to make the code look simpler

{TransportManager *tm_impl = MyTransportManagerImpl::instance();
}
2. Two queues are used as a fundamental of the TM
   - Message Queue for commands coming from the upper level
   - Event Queue for events coming from devices

3. Customizing the TM, the Developer
   - Should implement transport manager interface
   - Should use the transport adapter interface and the transport listener interface for making the implementation work with the default adapters and listeners
   - Has two options:
     - Creating the TM from scratch
     - Deriving from the default implementation
       - If not particularly changed, the default Adapters and Listeners will be used.

Transport Adapter Customizing

1. Basic information
   - TA is highly adaptable to any specific of a real transport
   - TA consists of
     - Worker Classes that perform a single operation (e.g., device search)
     - Controller that
       - Accumulates event handling from all worker classes
       - Controls the state of all internal data and
       - Notifies the upper level via callbacks
   - Internal data structures that contain the information about the device, the connection and other necessary information

2. Workers of TA
   - Device Scanner:
     - Implements transport dependent search procedure initiated by the appropriate command from the Transport Manager
     - It is the developer's responsibility to implement this worker
- It may be absent for transport types that do not support searching.
- When the device is found the worker:
  - Creates a notification and directs this notification to the Controller
  - The Controller notifies TM using the TA Listener
  - TM receives the notification and sends a command to TA for connecting all available applications
  - TM and TA perform a chain of notifications
  - TM notifies the upper level using TM Listener with the information on each application connected; the connection ID, the application name, the device name

- Client Connection Listener:
  - Implements the transport dependent connection that was originated by the device
  - It is the developer’s responsibility to implement this worker
  - If the transport does not support such ability the worker may be absent
  - Working procedure:
    - The worker waits for the connection of a mobile device
    - When the connection request from any of the mobile devices arrives, TA establishes a connection (creates the data path) with this device and the connection listener sends a notification through the controller to the upper level with the device and application IDS

- Server Connection Factory
  - Implements transport dependent connection that was originated by the user
  - It is the developer’s responsibility to implement this worker
  - If the transport does not support this ability, the worker may be absent
  - Creates a connection with the device and the application specified by the user
  - Both the device and the application must already be known to the TA at this moment
TA may know about the device after the 'search' routine or after the 'restore previous state' routine
- If device is not known the Transport Adapter returns the error immediately
  - When the connection is created the TA sends a notifications through the controller to the upper level

3. Connection

- The main responsibility of the client and server connection works is to create a Connection
- Connection is the entity that is responsible for data transmitting between Core and the device
- Works and Connection are closely related
- Customing the connection
  - Custom implementation of Connection must be used in custom worker(s) only
  - It is not possible to use other types of data exchange in default workers
  - It is possible to use the default Connection implementation in custom workers
  - The default Connection implementation is based on sockets isolated in a separate thread
  - Each transport specific worker shall use transport dependent initialization of the threaded socket connection
  - If the default implementation is not convenient the developer can create his own Connection
  - In this case custom workers shall also be created

4. The Descriptor is used for manipulating devices and connections inside of the adapter

CUSTOM TRANSPORT ADAPTER CREATION

Create a class that is derived from `TransportAdapterImpl` and add the implementation of necessary virtual methods.
Create a connection class deriving \texttt{ThreadedSocketConnection} and implement virtual methods

```cpp
#include "transport_adapter_impl.h"

class MyTransportAdapter : public TransportAdapterImpl
{
protected:
    virtual DeviceType getDeviceType() const;
};
```

Create a class for the device that will be used by Controller to manage devices and implement all virtual methods

```cpp
#include "transport_adapter_impl.h"

class MySocketConnection : public ThreadedSocketConnection
{
    virtual ~MySocketConnection();
protected:
    virtual bool establish(ConnectError **error);
};
```

```cpp
#include "transport_adapter_impl.h"

class MyDevice : public Device
{
    virtual ~Device();
    virtual bool isSameAs(const Device *other_device) const;
    virtual ApplicationList getApplicationList() const;
};
```
Create the necessary worker classes by deriving the appropriate basic worker and fill in the necessary functionality

```cpp
#include "transport_adapter_impl.h"

class MyDeviceScanner : public DeviceScanner
{
}

class MyServerConnectionFactory : public ServerConnectionFactory
{
}

class MyClientListener : public ClientConnectionListener
{
}

#include "transport_adapter_impl.h"

MyTransportAdapter::MyTransportAdapter() : TransportAdapterImpl(
    new MyDeviceScanner(),
    new MyServerConnectionFactory(),
    new MyClientListener()
)
{
}
```

These workers should use Connection and Device which were created in previous steps. For instance, scanner should add a list of devices to controller. This list will be used later when the `connect` request is received. To create a data path the connection class should be used. When the connection starts it will update Controller with the pointer to this connection.

When everything is created it is time to combine everything.
Create an instance of the new adapter and provide it to the Transport Manager

```cpp
#include "transport_manager.h"
#include "transport_manager_impl.h"
#include "my_transport_adapter_impl.h"
{
    TransportAdapter *my_transport_adapter = static_cast<
        TransportAdapter *>(new MyTransportAdapter);
    TransportManager *tm_impl = TransportManagerDefault::
        instance();
    tm_impl->addTransportAdapter(my_transport_adapter);
}
```

This adapter will use the default connection implementation and default Adapter Listener but three worker classes will implement developer's logic. Also, the Transport Adapter will provide device type in the developer's own way. The `TransportAdapterImpl` virtual (but not pure) methods `Store()` and `Restore()` can be reimplemented to provide resumption mechanism. Default implementations for both methods do nothing.

### SDL Core Video Stream Setup

### Initial Configuration

### Install Packages

```bash
sudo apt-get install git cmake build-essential libavahi-client-dev libsqlite3-dev chromium-browser libssl-dev libudev-dev libgtest-dev libbluetooth3 libbluetooth-dev bluez-tools gstreamer1.0* libpulse-dev
```
Clone the SDL Core Repository

Clone the SDL Core repository

```bash
sudo apt-get update
sudo apt-get upgrade
sudo ldconfig
```

CD into sdl_core/ and checkout the master branch

```bash
git clone https://github.com/smartdevicelink/sdl_core.git
```

Clone the SDL HMI Repository

Clone the Web HMI repository

```bash
git clone https://github.com/smartdevicelink/sdl_hmi.git
```
CD into sdl_hmi/ and checkout the master branch

```bash
sudo git checkout -b release/4.0.0 origin/master
```

**Setup the Build Environment**

Create build folder outside of sdl_core/ directory

```bash
mkdir build
cd build
```

Run CMAKE and install application

```bash
cmake ..../sdl_core
make
make install
```

Copy keys from bin/ to src/appMain

```bash
cp bin/myKey.pem src/appMain
cp bin/myCert.pem src/appMain
```
Pipe Stream Setup

GSTREAMER Setup

First, we must determine what gstreamer command works in your environment.

Start by finding a raw h.264 file (Example) and determine which of the these gstreamer commands successfully plays the example video:

- Ubuntu 14.04+

```bash
gst-launch-1.0 filesrc location=/path/to/h264/file !
    decodebin !
    videoconvert !
    ximagesink sync=false
```

or

```bash
gst-launch-1.0 filesrc location=/path/to/h264/file !
    decodebin !
    videoconvert !
    ximagesink sync=false
```

If you're using tcp, you can connect the stream directly with your phones ip address using
Make configuration changes to smartDeviceLink.ini

In the build folder directory, open src/appMain/smartDeviceLink.ini in a text editor and make the following change:

Under [MEDIA MANAGER] comment out the socket stream options and remove semicolon from the pipe stream options:

```ini
;VideoStreamConsumer = socket
;AudioStreamConsumer = socket
;VideoStreamConsumer = file
;AudioStreamConsumer = file
VideoStreamConsumer = pipe
AudioStreamConsumer = pipe
```

Prerolling Pipeline

After you start SDL Core, cd into the src/appMain/storage directory and there should be a file named "video_stream_pipe". Use the gst-launch command that:

```
gst-launch-1.0 tcpclientsrc host=<Device IP Address> port=3000 ! decodebin ! videoconvert ! ximagesink sync=false

* Audio Stream Pipe (Raw PCM)

```
```
gst-launch-1.0 filesrc location=${SDL_BUILD_PATH}/src/appmain/storage/audio_stream_pipe ! audio/x-raw,format=S32BE,rate=8000,channels=1 ! pulsesink
```

```
worked for your environment and set file source to the video_stream_pipe file. You should see “setting pipeline to PAUSED” and “Pipeline is PREROLLING”.

```
 gst-launch-1.0 filesrc location=$SDL_BUILD_PATH/src/appmain/storage/video_stream_pipe ! decodebin ! videoconvert ! xvimagesink sync=false
```

Start SDL Core

In the build folder

```
 cd src/appMain
 ./smartDeviceLinkCore
```

NOTE: If you want to use a USB connection, you must run

```
 sudo ./smartDeviceLinkCore
```

Start the web HMI

CD into the HMI repository and run

```
 chromium-browser index.html
```
Start Video Stream

To Do: Provide public mobile application that supports video streaming.

Mobile application settings (Wifi/TCP connection):

- Deselect Heartbeat.
- Select MobileNavi (video source).
- Select H264 video format.
- Select wifi as the connection type.
- Input the Virtual Machine's IP Address and port 12345.
- Press the Ok button in the App to start the connection.
- In the web HMI, click the italic "i" and select your app.
- In the app, select "start service" to request permission to stream video.
- The HMI will prompt you to give permission to the app to stream video.
  Click "Ok"
- In the app, press "Start File Streaming". Depending on your video
  playback configuration, the video will begin playing in the web browser, or
  a gstreamer window open and begin playing the video.

Mobile application settings (USB connection):

- After core is started, connect a phone to your machine with a usb cable.
- The phone will prompt which app you want to run, select your app.
- Start the app on your phone.
- Deselect Heartbeat.
- Select MobileNavi (video source).
- Select H264 video format.
- Select usb as the connection type.
- Press the Ok button in the App to start the connection.
- In the web HMI, click the italic "i" and select your app.
- In the app, select "start service" to request permission to stream video.
- The HMI will prompt you to give permission to the app to stream video.
  Click "Ok"
- In the app, press "Start File Streaming". Depending on your video
  playback configuration, the video will begin playing in the web browser, or
  a gstreamer window open and begin playing the video.
SmartDeviceLink

Release Notes (Release 4.2.0)

1. Introduction

Definitions and Abbreviations

<table>
<thead>
<tr>
<th>TERM</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CY</td>
<td>Calendar Year</td>
</tr>
<tr>
<td>CRQ</td>
<td>Change request</td>
</tr>
<tr>
<td>SDL</td>
<td>SmartDeviceLink</td>
</tr>
<tr>
<td>ATF</td>
<td>Automated Test Framework</td>
</tr>
<tr>
<td>GitHub</td>
<td>Source code revision system with released version of OpenSDL</td>
</tr>
</tbody>
</table>
Scope

App Launch (iOS):

Integration of functionality already implemented in F-S SDL. Within the scope of the CRQ integration, SDL team removed iAP2 transport implementation and Multiplexing functionality available in F-S SDL.

New implementation of requested functionality:

- Navigation interface: SDL behavior in case HMI does not respond to IsReady_request or respond with "available" = false
- TTS interface: SDL behavior in case HMI does not respond to IsReady_request or respond with "available" = false
- UI interface: SDL behavior in case HMI does not respond to IsReady_request or respond with "available" = false
- VR interface: SDL behavior in case HMI does not respond to IsReady_request or respond with "available" = false
- VehicleInfo interface: SDL behavior in case HMI does not respond to IsReady_request or respond with "available" = false

2 About This Release

Implemented functionality for remote launching the applications from the supported Launch function devices.
Note: It is a business logic without supported device implementation.
For further details, please refer iAP, AOA, SDL Core SAD or your transport API documentation.

Changed SDL behavior in case HMI does not respond to IsReady_request or respond with "available" = false for following interfaces: VR, UI, TTS, Navigation and VehicleInfo interfaces.
Short description of new behavior (*INTERFACE* is generic term used to described any of VR, UI, TTS, Navigation and VehicleInfo interfaces):

1. HMI respond *INTERFACE*.IsReady (false) -> SDL must return 'UNSUPPORTEDRESOURCE, success:false' to all single *INTERFACE*-related RPC
2. HMI respond *INTERFACE*.IsReady (false) and app sends RPC that must be splitted -> SDL must NOT transfer *INTERFACE* portion of splitted RPC to HMI
3. HMI does NOT respond to *INTERFACE*.IsReady_request -> SDL must transfer received RPC to HMI even to non-responded *INTERFACE* module

3 Environment and dependencies

Development and testing environment for OpenSDL Ubuntu 14.04 LTS x32/x64

- *Debug Environment*: Ubuntu 14.04 LTS x32/x64, Qt 5.3
- *Compiler*: GCC 4.9.3 (OS Ubuntu), Lua 5.2
- *Build system*: Cmake 2.8.12.2

Development and testing environment for OpenSDL Windows x64:

- *Build system*: Windows 7 x64, CMake
- *Compiler*: Microsoft Visual Studio Express Edition 2013 x64
- *Development and testing environment for OpenSDL Qt for Windows x32*:
  - *Build system*: Windows 7 x32, Qt 5.5, CMake, QT Creator
  - *Compiler*: Microsoft Visual Studio Express Edition 2010 x32
Source Control System:

- GitHub

4. Delivery details

Unit Tests Coverage

<table>
<thead>
<tr>
<th>COVERAGE</th>
<th>HIT</th>
<th>TOTAL</th>
<th>COVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines</td>
<td>18153</td>
<td>27828</td>
<td>66 %</td>
</tr>
<tr>
<td>Functions</td>
<td>7646</td>
<td>11830</td>
<td>65 %</td>
</tr>
</tbody>
</table>

Tests Execution Report

<table>
<thead>
<tr>
<th>TESTS TOTAL</th>
<th>FAILURES TOTAL</th>
<th>DISABLED TOTAL</th>
<th>ERRORS TOTAL</th>
<th>TOTAL TIME (MILLISECONDS)</th>
<th>TESTS TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1748</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>310482</td>
<td>1748</td>
</tr>
</tbody>
</table>

Brief log of unit test sets execution:
01/26 Test #01: test_JSONCPP ....................... Passed 0.18 sec
02/26 Test #02: test_generated_interface .......... Passed 0.09 sec
03/26 Test #03: transport_manager_test .......... Passed 1.63 sec
04/26 Test #04: resumption_test ................... Passed 0.06 sec
05/26 Test #05: formatters_test ................... Passed 0.73 sec
06/26 Test #06: protocol_handler_test .......... Passed 22.37 sec
07/26 Test #07: connection_handler_test .......... Passed 83.31 sec
08/26 Test #08: utils_test ........................ Passed 36.30 sec
09/26 Test #09: generator_test .................... Passed 0.08 sec
10/26 Test #10: security_manager_test .......... Passed 10.28 sec
11/26 Test #11: policy_test ....................... Passed 108.36 sec
12/26 Test #12: rpc_base_test ..................... Passed 0.18 sec
13/26 Test #13: smart_object_test ................. Passed 2.23 sec
14/26 Test #14: application_manager_test ........ Passed 3.00 sec
15/26 Test #15: resumption/data_resumption_test ... Passed 0.37 sec
16/26 Test #16: state_controller_test .......... Passed 0.37 sec
17/26 Test #17: app_launch_ctrl_test .......... Passed 43.12 sec
18/26 Test #18: app_launch_data_test .......... Passed 0.04 sec
19/26 Test #19: commands_test ..................... Passed 0.05 sec
20/26 Test #20: mobile_commands_test .......... Passed 0.19 sec
21/26 Test #21: hmi_commands_test ................. Passed 0.05 sec
22/26 Test #22: message_helper_test ............... Passed 0.01 sec
23/26 Test #23: hmi_message_handler_test .......... Passed 0.02 sec
24/26 Test #24: config_profile_test ............... Passed 0.33 sec
25/26 Test #25: media_manager_test ............... Passed 0.02 sec
26/26 Test #26: telemetry_monitor_test .......... Passed 0.02 sec
100% tests passed

5. Known Bugs and Limitations

All known SDL defects reflected in following chapter. The correction and verification of those defects are out of scope of this release.
Known Issues
<table>
<thead>
<tr>
<th>SUMMARY</th>
<th>PRIORITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Genivi] SDL doesn't stop at execution ATF function StopSDL()</td>
<td>Blocker</td>
</tr>
<tr>
<td>[Genivi]: Core crash upon Ctrl+C in console</td>
<td>Critical</td>
</tr>
<tr>
<td>[Genivi][Policies] PTU is not successful due to another unexpected exchange in progress</td>
<td>Critical</td>
</tr>
<tr>
<td>[Genivi] SDL stops working during processing SetGlobalProperties request</td>
<td>Critical</td>
</tr>
<tr>
<td>[SDL4.0][Genivi] SDL sends OnSystemRequest(QUERY_APPS) to background on phone App.</td>
<td>Critical</td>
</tr>
<tr>
<td>[Genivi] [TM] Unable to register iOS App via BT.</td>
<td>Critical</td>
</tr>
<tr>
<td>[Genivi][Security] SDL do not send certificate from Policy DB and rewrites certificate in module_config with &quot;1&quot; right after using it</td>
<td>Critical</td>
</tr>
<tr>
<td>[Genivi][Security] SDL crashes if App tries to restore secure RPC service on start</td>
<td>Critical</td>
</tr>
<tr>
<td>[Genivi][Security] SDL crashes if during TLS handshake ERROR_SSL_INVALID_DATA occurs</td>
<td>Critical</td>
</tr>
<tr>
<td>Summary</td>
<td>Priority</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>[GENIVI][WinQT] 3rd party USB library crash on exit</td>
<td>Critical</td>
</tr>
<tr>
<td>[Resumption][Genivi] SDL crashes during resumption of 2 Apps, non-media to FULL and media to LIMITED</td>
<td>Critical</td>
</tr>
<tr>
<td>[GENIVI] SDL should respond &quot;IGNORED&quot; with correct result code for UnSubscribeVehicleData in case vi interface isn't available</td>
<td>Critical</td>
</tr>
<tr>
<td>[GENIVI][Policy]: SDL does not send RequestType:HTTP in OnSystemRequest to app</td>
<td>Critical</td>
</tr>
<tr>
<td>APPLINK-17839 Genivi: HMILevel is not resumed to LIMITED for non-media applications</td>
<td>Major</td>
</tr>
<tr>
<td>APPLINK-17839 Genivi: HMILevel resumption is not canceled at OnEventChanged (AUDIO_SOURCE, isActive: true)</td>
<td>Major</td>
</tr>
<tr>
<td>[SDL4.0][Genivi] UTF-8: Core incorrect handles symbols of two or more bytes size</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi][Policies] SDL does not send OnPermissionsChange after PTU</td>
<td>Major</td>
</tr>
<tr>
<td>SUMMARY</td>
<td>PRIORITY</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>[Genivi][Policies] SDL doesn't exclude messages from snapshot</td>
<td>Major</td>
</tr>
<tr>
<td>[GENIVI][Policy]: SDL dos not select url from PT for specified appID during GetURLs request.</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi] Policies Manager does not revert the Local Policy Table to the Preload Policy Table upon FACTORY Reset</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi][Policies] SDL doesn't send &quot;SDL.OnAppPermissionChanged {appID} to HMI</td>
<td>Major</td>
</tr>
<tr>
<td>[GENIVI][Security]: App continue unprotected stream if start service as protected during active streaming</td>
<td>Major</td>
</tr>
<tr>
<td>[GENIVI][Memory leaks]: SDL does not release memory after sending AddCommand limit exhausted</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi]: SDL send to mobile &quot;APPLICATION_NOT_REGISTRED&quot; in setAppIcon response if HMI respond with &quot;INVALID_DATA&quot;</td>
<td>Major</td>
</tr>
<tr>
<td>[GENIVI] Incorrect response on send SystemRequest (file name - /test)</td>
<td>Major</td>
</tr>
<tr>
<td>[GENIVI] putFile does not copy file with \ before the name to AppStorageFolder</td>
<td>Major</td>
</tr>
<tr>
<td>SUMMARY</td>
<td>PRIORITY</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>[GENIVI]SDL retry send StartStream/StartAudioStream less on one time than configured in .ini file</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi][Policies] SDL doesn't reject PT if the consumer_friendly_message section contains messaging without “en-us” language key</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi][Policies] PM should verify that &quot;seconds_between_retries&quot; array has maxlen 5</td>
<td>Major</td>
</tr>
<tr>
<td>[GENIVI] SDL transfer OnKeyboardInput notification to not active App when there is no active PerformInteraction (KEYBOARD)</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi] SDL forwards OnButtonPress (CUSTOM_BUTTON) with wrong appID to current App.</td>
<td>Major</td>
</tr>
<tr>
<td>GENIVI: SDL doesn't send &quot;REJECTED&quot; code to mobile app when activating app from HMI with activate Carplay/GAL.GE</td>
<td>Major</td>
</tr>
<tr>
<td>SUMMARY</td>
<td>PRIORITY</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>GENIVI: App is disconnected due to HeartBeat timeout although HeartBeat is sent.</td>
<td>Major</td>
</tr>
<tr>
<td>Genivi SDL blocks forever when registering mobile application with Genivi HMI (only)</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi] HMI level resumption is not postponed at EmergencyEvent, isActive=true</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi] SDL doesn't apply sequence SUSPEND -&gt; OnSDLAwake -&gt; SUSPEND -&gt; IGN_OFF for saving resumption data.</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi][API] SDL sends UpdateDeviceList with disconnected device in the deviceList</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi][API] SDL sends appName in vrSynonyms and ttsName in case of lower and upper bound values of params in json file</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi][API] App is not unregistered by reason = REQUEST_WHILE_IN_NONE_HMI_LEVEL</td>
<td>Major</td>
</tr>
<tr>
<td>SUMMARY</td>
<td>PRIORITY</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>[Genivi][API] SDL sends OnApplInterfaceUnregistered (\text{DRIVER_DISTRACTION_VIOLATION}) to app when receives OnExitApplication (\text{DRIVER_DISTRACTION_VIOLATION}) from HMI</td>
<td>Major</td>
</tr>
<tr>
<td>[GENIVI] One and the Same Correlation_ID is assigned by SDL Two Times</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi][SDL4.0] SDL does not send OnSystemRequest to app on second device</td>
<td>Major</td>
</tr>
<tr>
<td>Genivi: Policy table can't be loaded when RPCs added in functional_group is greater than 50.</td>
<td>Major</td>
</tr>
<tr>
<td>[GENIVI][Policy]: SDL does not write UserFriendlyMessages to DB</td>
<td>Major</td>
</tr>
<tr>
<td>GENIVI: PerformAudioThru - SDL does not send &quot;resultCode:RETRY, success:true&quot; to mobile app when press &quot;Retry&quot; button</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi][Policy] PM doesn't validate required section/key in case it is invalid and SDL continue running</td>
<td>Major</td>
</tr>
<tr>
<td>In Genivi (SDL 4) we can have two mobile apps in FULL HMI level at the same time</td>
<td>Major</td>
</tr>
<tr>
<td>SUMMARY</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td></td>
</tr>
<tr>
<td>[Genivi][Policies] Ford-specific keys are present in Genivi Policy DB - usage_and_error_counts</td>
<td></td>
</tr>
<tr>
<td>[Genivi][Policies] PM doesn't update &quot;notifications_per_minute_by_priority&quot;</td>
<td></td>
</tr>
<tr>
<td>[Genivi][Policies] OnSystemRequest: SDL does not re-send OnSystemRequest notification to mobile app in case when it was sent from HMI without appID</td>
<td></td>
</tr>
<tr>
<td>Unable to build GENIVI SDL without logs</td>
<td></td>
</tr>
<tr>
<td>[Genivi] SDL do not apply nicknames after PTU</td>
<td></td>
</tr>
<tr>
<td>[Genivi] SDL creates redundant device_consent table in Policy DB</td>
<td></td>
</tr>
<tr>
<td>[Genivi][Protocol] App becomes unregistered if PutFile is sent from any of two sessions (protocols v.2 and v.3)</td>
<td></td>
</tr>
<tr>
<td>[Genivi][Protocol] SDL respond ACK with protocol version 4 for video and audio services if send start service with 2 or 3 protocol version</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PRIORITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major</td>
</tr>
<tr>
<td>Major</td>
</tr>
<tr>
<td>Major</td>
</tr>
<tr>
<td>Major</td>
</tr>
<tr>
<td>Major</td>
</tr>
<tr>
<td>Major</td>
</tr>
<tr>
<td>Major</td>
</tr>
<tr>
<td>Major</td>
</tr>
<tr>
<td>Major</td>
</tr>
<tr>
<td>SUMMARY</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>[Genivi][Policies] SDL should be case-insensetive to &quot;AppID&quot; against listed in policies manager</td>
</tr>
<tr>
<td>[Genivi][Security] SDL close connection before UNSUPPORTED_VERSION response for RAI was sent.</td>
</tr>
<tr>
<td>[Genivi][Policies] PT is considered as valid with no items in &quot;groups&quot; sub-section from &quot;default&quot;</td>
</tr>
<tr>
<td>[Genivi][TV] SDL must respond with INVALID_DATA on SystemRequest that uploads a file containing <code>../</code> sequences</td>
</tr>
<tr>
<td>[Genivi][IVSU] SDL doesn't reject SystemRequest with filenam=IVSU but w/o binary data.</td>
</tr>
<tr>
<td>[Genivi] Core dump upon FACTORY_DEFAULT</td>
</tr>
<tr>
<td>[Genivi][Policies] PM doesn't validate the size of section &quot;default&quot; in &quot;endponts&quot; of Policy Table</td>
</tr>
<tr>
<td>[Genivi][Policy]: PM doesn't merge &quot;functional_grouping&quot; and &quot;message_type&quot;</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>[Genivi][APIs] AlertManeuver: SDL responds GENERIC_ERROR instead of INVALID_DATA when soft button has Type is Image or Both and Text is whitespace or \t or \n or empty</td>
</tr>
<tr>
<td>[Genivi][API] AlertManeuver: SDL responds to mobile app UNSUPPORTED_REQUEST with success = true</td>
</tr>
<tr>
<td>[Genivi][SDL4.0] Json validation is failed in case language parameter does not contain vrSynonyms or ttsName</td>
</tr>
<tr>
<td>[Genivi][SDL4.0] SDL sends OnSystemRequest(QUERY_APPS) to the app after unsuccessful attempt</td>
</tr>
<tr>
<td>[Genivi] SDL returns IGNORED instead of UNSUPPORTEDRESOURCE for UnsubscribeButton.</td>
</tr>
<tr>
<td>[Genivi] SDL do not send default vrHelp to HMI if App was registered with vrSynonyms</td>
</tr>
<tr>
<td>[Genivi] [TM] SDL can't reregister App via USB that was killed before.</td>
</tr>
<tr>
<td>[Genivi] OnHashChange notification for UnsubscribeVehicleData when 2 applications are registered</td>
</tr>
<tr>
<td><strong>SUMMARY</strong></td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GENIVI: SDL responds &quot;resultCode: SUCCESS&quot; while dataType: VEHICLEDATA_EXTERN TEMP is VEHICLE_DATA_NOT_AVAILABLE and not in subscribed list store</td>
</tr>
<tr>
<td>[GENIVI] No response to App on UI.Slider sent if no HMI response during DefaultTimeout</td>
</tr>
<tr>
<td>[GENIVI][Policy]: SDL does not set &quot;timeout&quot; for OnSystemRequest with url</td>
</tr>
<tr>
<td>[GENIVI][Policy]: SDL does not apply url from PT for specified appID for OnSystemRequest</td>
</tr>
<tr>
<td>[Genivi] INVALID_DATA received in case wayPointType with correct parameter was sent</td>
</tr>
<tr>
<td>[Genivi][API] SDL responds &quot;UNSUPPORTED_RESOURCE&quot;, success= false in case only have &quot;UNSUPPORTED_RESOURCE&quot; to Navigation.AlertManeuver</td>
</tr>
<tr>
<td>GENIVI: App is disconnected due to PROTOCOL_VIOLATION when start audio streaming after rejected 2 times then accepted.</td>
</tr>
<tr>
<td>[GENIVI]: SDL crashes during execution ATF test with start-stop SDL cases</td>
</tr>
<tr>
<td>Summary</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>[GENIVI] Redundant info is sent to App when single UI. got 'UNSUPPORTEDRESOURCE' from HMI</td>
</tr>
<tr>
<td>[Genivi] SDL responses with GENERIC_ERROR instead of UNSUPPORTEDRESOURCE</td>
</tr>
<tr>
<td>[GENIVI] SDL does not transfer all info parameters of unsuccess result codes when UI.IsReady = false</td>
</tr>
<tr>
<td>[GENIVI] SDL does not respond info message in case GENERIC_ERROR watchdog timeout from HM</td>
</tr>
<tr>
<td>[GENIVI][Navigation] SDL does not respond info message in case GENERIC_ERROR watchdog timeout from HM</td>
</tr>
<tr>
<td>[GENIVI] GetWayPoints: SDL does not reset the default watchdog timeout of GetWayPoints request if HMI sends OnResetTimeout notification for this request</td>
</tr>
<tr>
<td>[GENIVI] HashID should not be updated on successful single UI. if no UI.IsReady response</td>
</tr>
<tr>
<td><strong>SUMMARY</strong></td>
</tr>
<tr>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>[GENIVI] USER_DISALLOWED send instead of IGNORED for UnsubscribedVehicleData when VehicleInfo IsReady Missing</td>
</tr>
<tr>
<td>[GENIVI] SDL responds &quot;WARNINGS&quot; code in case SDL got &quot;WARNINGS&quot; from TTS and error code from other interfaces.</td>
</tr>
<tr>
<td>[GENIVI] SDL doesn't set unsuccessful &quot;message&quot; value to &quot;info&quot; param in case HMI responds via single UI.RPC when .IsReady missing</td>
</tr>
<tr>
<td>[Policies][Genivi] PM shuts SDL down but doesn't log error in case any required section/key is absent in the sdl_preloaded_pt.json</td>
</tr>
<tr>
<td>GENIVI: SDL does not send StopAudioStream() if exit app while Video service and Audio service are starting.</td>
</tr>
<tr>
<td>[GENIVI] The session registration is delayed by locks</td>
</tr>
<tr>
<td>[Genivi][SDL4.0] SDL does not create icons folder in case it was removed after SDL start</td>
</tr>
<tr>
<td>Summary</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>[Genivi][API] SDL sends BC.UpdateDeviceList with out of upper bound size of deviceList</td>
</tr>
<tr>
<td>[Genivi][SDL4.0] SDL sends wrong parameter names in OnSystemRequest(query_apps)</td>
</tr>
<tr>
<td>[GENIVI][Policy]: SDL increments &quot;ignition_cycles_since_last_exchange&quot; counter after ign cycle if PTU was not before</td>
</tr>
<tr>
<td>[SDL4.0][Genivi] incorrect resp. in case Unsubscribe not supported and not yet subscribed button</td>
</tr>
<tr>
<td>[Genivi][Policies] Ford-specific keys are present in Genivi Policy DB - DeviceData</td>
</tr>
<tr>
<td>[Genivi][Policies]: &quot;application&quot; table contains &quot;certificate&quot;</td>
</tr>
<tr>
<td>[Genivi] SDL doesn't add device identifier to Policy DB and not present in the snapshot</td>
</tr>
<tr>
<td>[Genivi][Security] SDL print out CN and serialNumber details of certificates in log.</td>
</tr>
<tr>
<td>[Genivi][VS][AS] SDL does not send StopStream/StopAudioStream to HMI after unregistering app during streaming</td>
</tr>
<tr>
<td>SUMMARY</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>[Genivi] Communication is not saved in SmartDeviceLinkCore.log after adding persistent data after SUSPEND</td>
</tr>
<tr>
<td>[Genivi] SDL doesn't transfer info parameter with UI.SetGlobalProperties to mobile.</td>
</tr>
<tr>
<td>[Genivi] SDL sends systemSoftwareVersion (with empty value) in RAI response if before ccpu_version was invalid in GetSystemInfo.</td>
</tr>
<tr>
<td>[Genivi] [Policies] Policy table is not initialized by SDL start without DB</td>
</tr>
<tr>
<td>[GENIVI] Response to UI.ScrollableMessage is sent twice to app if no HMI response HMI during</td>
</tr>
<tr>
<td>[Genivi] SDL doesn't send info parameter when result of ResetGlobalProperties is GENERIC_ERROR</td>
</tr>
<tr>
<td>Genivi: UI is waiting for TTS.IsReady timeout to elapse to send UI.</td>
</tr>
<tr>
<td>[Genivi] SDL response success:false to mobile app in case it received RETRY or WRONG_LANGUAGE or UNSUPPORTEDRESOURCE from HMI</td>
</tr>
<tr>
<td>Summary</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>GENIVI: SDL always responds &quot;INVALID_DATA&quot; to mobile app while receiving other code from HMI</td>
</tr>
<tr>
<td>[Genivi] Incorrect Version of API in MOBILE_API.xml</td>
</tr>
<tr>
<td>[GENIVI] SDL build without any error message with empty version in MOBILE_API.xml</td>
</tr>
<tr>
<td>[GENIVI] SyncMsgVersion is not updated after RegisterAppInterface</td>
</tr>
</tbody>
</table>