Core Documentation

In the Core documentation you will find information about the architecture of the SDL embedded component (SDL Core), as well as information on how to customize this application to work well with your vehicle.

First of all, the SDL Core SW Architecture document provides an overview the SDL Core application as well as the SDL platform.

For Integration purpose, please follow:

- Deployment schema
- Operational aspects
  - Configuration
  - Logging
  - Diagnostics
- Preloaded Policy Table configuration

For Testing purpose, please follow:

- Logging and diagnostics
- Automated Test Framework (ATF) SW Architecture
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.
### Definitions used in this document are in the table below.

<table>
<thead>
<tr>
<th><strong>DEFINITION</strong></th>
<th><strong>DESCRIPTION</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Concern</strong></td>
<td>A functional or non-functional requirement.</td>
</tr>
<tr>
<td><strong>Model</strong></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><strong>Stakeholder</strong></td>
<td>An individual, group or organization that has at least one concern relating to the system.</td>
</tr>
</tbody>
</table>
1.3. Document Roadmap

The SW architecture of system is considered from different viewpoints:
<table>
<thead>
<tr>
<th>VIEWPOINT</th>
<th>VIEWPOINT DESCRIPTION</th>
</tr>
</thead>
<tbody>
<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>Data</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>Process State</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>Process</td>
<td>Concurrency type of view. Process View describes processes and process inter-communication mechanisms independent of physical hardware deployment.</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>
</tbody>
</table>
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.
4. Views

4.1. Use Case View

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

Transfer data

Transfer Control data

Transfer RPC data

Transfer bulk data

Transfer Audio data

Transfer Video data

Establish RPC service

Establish bulk service

Establish Audio service

Establish Video service

Establish Session

Control service data could be not related to any session
DATA VERIFICATION USE CASE DIAGRAM

Mobile application

Transfer data

Check malformed

Application Unregistration

HMI / Applink

Transfer RPC

Check flood

Validate Data structure

Check Policy permission

Process HMI and internal errors
RPC USE CASE DIAGRAM

- Mobile application
- Send Mobile RPC
- Receive Mobile RPC
- Transfer RPC
- Receive HMI RPC
- Send HMI RPC
- HMI / Applink Interface
MOBILE TO HMI RPC PROCESSING USE CASE DIAGRAM
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-specifcs 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.

**PLUGIN MANAGER**

• **Responsibility**
  - Loads all .so files from specific directory, checking if they’re exporting required methods
  - Stores information about plugin capabilities
  - Checks plugins capability to handle RPCs

• **Relations**
  - Composed by *Application Manager*
  - Composes *Plugin*

• **Interfaces**
  - Provides *Plugin Manager* interface

• **Behavior**
  - Loads and manages plugins from specific directory.

• **Constraints**
  - Able to load only RPC layer plugins

**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

REMOTE CONTROL

• **Responsibility**
  ◦ Allows incorporating additional functionality to the core application by application extension.
  ◦ Implements specific mobile RPC handling.
  ◦ Implements specific HMI RPC handling.

• **Relations**
  ◦ Composed by *Plugin manager*
• Handles Application Manager by Service interface

• Interfaces
  ◦ Provides Plugin Manager interface

• Behavior
  ◦ Receives data from CoreService
  ◦ Parses data
  ◦ Creates commands.
  ◦ Handles incoming HMI notifications
  ◦ Sends RPC to HMI
  ◦ Sends RPC to mobile
  ◦ Extends basic applications with additional RPC's

• Constraints
  ◦ N/A

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 TransportAdapters according to configuration

• Interfaces
  ◦ Provides TransportManager interface for devices and connections status manipulation
  ◦ Provides TransportManagerListener interface for transport notification subscribing

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

• Constraints
  ◦ N/A

TRANSPORT ADAPTER

• Responsibility
• Transport-specific API encapsulation
• Relations
  ◦ Composed by TransportManager
• Interfaces
  ◦ Provides **TransportAdapters** interface

• Behavior
  ◦ Adopts transport searching, connecting, data transferring API for one **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**
4.3.2. Protocol layer

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

Protocol Layer - data transferring diagram

View Diagram
4.3.3. Business layer

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

**SEQUENCE DIAGRAM**
Business layer - media data transferring diagram

Business layer - RPC processing diagram
SEQUENCE DIAGRAM

Business layer - App Launch

View Diagram

loop [For each saved app]

"Business layer - App Launch"

Connection Handler Application Manager App Launch Resumption

Connection Handler Application Manager App Launch Resumption

Notification

Device connected

Device Connected

Get apps data for device

Get HMI level for app

Launch app

Launch app

SEQUENCE DIAGRAM

Business layer - Plugin Manager

View Diagram
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.
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
  ◦ Plugin Manager
  ◦ 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
  - Build system: Cmake 2.8.12.2

- Development and testing environment for SDL Windows x64:
  - Build system: Windows 7 x64, CMake
  - Compiler: Microsoft Visual Studio Express Edition 2013 x64

- Development and testing environment for SDL Qt for Windows x32:
  - Build system: Windows 7 x32, Qt 5.5, CMake, QT Creator
  - Compiler: Microsoft Visual Studio Express Edition 2010 x32

- Requirements Management system: LuxProject (JIRA, Confluence)
- Source Control System: GitHub
- Issue Tracking System: LuxProject (JIRA)
- Document Management System: LuxProject (JIRA, Confluence, SVN)
- Coding style: *SDL C++ Style*

### 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
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></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Config Profile</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application Manager</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command s</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Request Controller</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>App Launch</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Resumption</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Policy</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Media Manager</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Protocol Handler</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Connection Handler</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Security Manager</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>HMI Message Handler</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Plugin Manager</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transport Manager</td>
<td></td>
<td></td>
<td>X</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>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>DATA / LAYER</th>
<th>APPLICATION LAYER</th>
<th>PROTOCOL LAYER</th>
<th>TRANSPORT LAYER</th>
<th>OS LAYER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SmartObject</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mobile Command</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HMI Command</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProtocolFrame</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SecurityQuery</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>RawMessage</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JSON::Value</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>DBUS message</td>
<td></td>
<td></td>
<td>X</td>
<td>X</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 append 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
issueuetype=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
issueuetype=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
RESUMPTION USE CASE DIAGRAM

APPLICATION DATA RESUMPTION USE CASE DIAGRAM

HMI LEVEL RESUMPTION USE CASE DIAGRAM

SOLUTION OVERVIEW

COMPONENT VIEW DIAGRAM

TRANSPORT LAYER NOTIFICATION AND DATA TRANSFERRING DIAGRAM

PROTOCOL LAYER - TRANSPORT NOTIFICATIONS PROCESSING DIAGRAM

PROTOCOL LAYER - DATA TRANSFERRING DIAGRAM

BUSINESS LAYER - MEDIA DATA TRANSFERRING DIAGRAM
BUSINESS LAYER - RPC PROCESSING DIAGRAM

DATA FLOW DIAGRAM

LIFE CYCLE STATES DIAGRAM

DEVELOPMENT VIEW DIAGRAM

DEPLOYMENT VIEW DIAGRAM

9. APPENDICES

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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Chapter 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'Views' creation</td>
</tr>
<tr>
<td>1.1</td>
<td>04/30/2013</td>
<td>Draft</td>
<td>Klimenko Dmitriy</td>
<td>Chapter 5</td>
</tr>
<tr>
<td>1.2</td>
<td>05/10/2013</td>
<td>Draft</td>
<td>Polina Vyshnevsk</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>for the Views</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>creation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Views diagrams and</td>
</tr>
<tr>
<td>1.4</td>
<td>05/27/2013</td>
<td>Draft</td>
<td>Alexandr Kandul</td>
<td>description</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>update</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Chapters 7,8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>creation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>formatting</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>grammar corrections</td>
</tr>
<tr>
<td>1.5</td>
<td>05/28/2013</td>
<td>Draft</td>
<td>Anastasiya Britanova</td>
<td>Ready for customer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>review</td>
</tr>
<tr>
<td>2.0</td>
<td>05/30/2013</td>
<td>Draft</td>
<td>Anastasiya Britanova</td>
<td>Changes in</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>accordance to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>implementation</td>
</tr>
<tr>
<td>2.1</td>
<td>09/30/2013</td>
<td>Draft</td>
<td>Polina Vyshnevsk</td>
<td>Reviewed with</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>customer</td>
</tr>
<tr>
<td>2.2</td>
<td>10/31/2013</td>
<td>Reviewed</td>
<td>Polina Vyshnevsk, Pavel Savyelyev</td>
<td>Policy component</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>updated</td>
</tr>
<tr>
<td>2.3</td>
<td>05/23/2014</td>
<td>Draft</td>
<td>Andrey Oleynik</td>
<td>Resumptio</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>n component</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>updated</td>
</tr>
<tr>
<td>2.4</td>
<td>01/30/2015</td>
<td>Draft</td>
<td>Aleksandr Kutsan</td>
<td>Update policy and</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SDL component s</td>
</tr>
<tr>
<td>2.5</td>
<td>02/16/2015</td>
<td>Draft</td>
<td>Alexander Galiuzov</td>
<td>Policy component</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>updated</td>
</tr>
<tr>
<td>2.6</td>
<td>03/24/2015</td>
<td>Draft</td>
<td>Andrey Oleynik</td>
<td>Ported to Luxoft</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Confluence</td>
</tr>
<tr>
<td>2.7</td>
<td>08/25/2015</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td></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></td>
<td></td>
<td></td>
<td></td>
<td>Deleted Logical VIEW</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Update Solution Overview, View-to-View Relations, Solution Background</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Update references list and Operation view with diagnostics and configuraton information</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Added Process State View, Process View, Component Interaction View</td>
</tr>
<tr>
<td>2.13</td>
<td>06/06/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td></td>
</tr>
<tr>
<td>2.14</td>
<td>06/17/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td></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>2.15</td>
<td>06/29/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>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 reference numbering Spelling fixes</td>
</tr>
<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>CHANGE 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 Component's 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 Use cases moved to a separate chapter Add application's data storing in Process State View Update Diagnostics chapter with log4cxx features usage</td>
</tr>
<tr>
<td>3.3</td>
<td>08/11/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td></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 Add architecture changes related to SDL RC</td>
</tr>
<tr>
<td>3.7</td>
<td>06/09/2017</td>
<td>Draft</td>
<td>Aleksandr Kutsan</td>
<td></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.

<table>
<thead>
<tr>
<th>VERSION</th>
<th>DATA</th>
<th>APPROVER</th>
<th>APPROVE ITEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2</td>
<td>06/11/2013</td>
<td>Julius Marchwicki</td>
<td>-</td>
</tr>
<tr>
<td>2.2</td>
<td>06/11/2013</td>
<td>Pavel Savyelyev</td>
<td>APPLINK-3967</td>
</tr>
<tr>
<td>3.0</td>
<td>06/30/2016</td>
<td>Nataly Snitsar</td>
<td>APPLINK-25883</td>
</tr>
<tr>
<td>3.0</td>
<td>08/01/2016</td>
<td>Justin Dickow</td>
<td>PR #4</td>
</tr>
</tbody>
</table>
**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

QUERYAPPSURL

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.
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>FAILURE 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.
<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>[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><strong>SUMMARY</strong></td>
<td><strong>PRIORITY</strong></td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>[Genivi][Policies] SDL does not send OnPermissionsChange after PTU</td>
<td>Major</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}&quot; 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 responce 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>[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 maxlength 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/GALGE</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]CreateInteractionChoiceSet: core successfully creates choice set with duplicate vrCommands/menuName inside it.</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi][SDL4.0]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>[Genivi][API] SDL sends OnAppInterfaceUnregistered (DRIVER DISTRACTION VIOLATION) to app when receives OnExitApplication (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>[Genivi][Policies] Ford-specific keys are present in Genivi Policy DB - usage_and_error_counts</td>
<td>Major</td>
</tr>
<tr>
<td>Summary</td>
<td>Priority</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>[Genivi][Policies] PM doesn't update &quot;notifications_per_minute_by_priority&quot;</td>
<td>Major</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>Major</td>
</tr>
<tr>
<td>Unable to build GENIVI SDL without logs</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi] SDL do not apply nicknames after PTU</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi] SDL do not disallow API of revoked App.</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi] SDL creates redundant device_consent table in Policy DB</td>
<td>Major</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>Major</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>Major</td>
</tr>
<tr>
<td>[Genivi][Policies] SDL should be case-insensetive to &quot;AppID&quot; against listed in policies manager</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi][Security] SDL close connection before UNSUPPORTED_VERSION response for RAI was sent.</td>
<td>Major</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>
<td>Major</td>
</tr>
<tr>
<td>[Genivi][TV] SDL must respond with INVALID_DATA on SystemRequest that uploads a file containing ../ sequences</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi][IVSU] SDL doesn't reject SystemRequest with filenam=IVSU but w/o binary data.</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi] Core dump upon FACTORY_DEFAULT</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi][Policies] PM doesn't validate the size of section &quot;default&quot; in &quot;endpoints&quot; of Policy Table</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi][Policy]: PM doesn't merge &quot;functional_grouping&quot; and &quot;message_type&quot;</td>
<td>Major</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>
<td>Major</td>
</tr>
<tr>
<td><strong>SUMMARY</strong></td>
<td><strong>PRIORITY</strong></td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>[Genivi][API]AlertManeuver: SDL responds to mobile app UNSUPPORTED_REQUEST with success = true</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi][SDL4.0] Json validation is failed in case language parameter does not contain vrSynonyms or ttsName</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi][SDL4.0] SDL sends OnSystemRequest (QUERY_APPS) to the app after unsuccessful attempt</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi] SDL returns IGNORED instead of UNSUPPORTED_RESOURCE for UnsubscribeButton.</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi] SDL do not send default vrHelp to HMI if App was registered with vrSynonyms</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi][TM] SDL can’t reregister App via USB that was killed before.</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi] OnHashChange notification for UnsubscribeVehicleData when 2 applications are registered</td>
<td>Major</td>
</tr>
<tr>
<td>GENIVI: SDL responds &quot;resultCode: SUCCESS&quot; while dataType: VEHICLEDATA_EXTERNTEMP is VEHICLE_DATA_NOT_AVAILABLE and not in subscribed list store</td>
<td>Major</td>
</tr>
<tr>
<td>[GENIVI] No response to App on UI.Slider sent if no HMI response during DefaultTimeout</td>
<td>Major</td>
</tr>
<tr>
<td>[GENIVI][Policy]: SDL does not set &quot;timeout&quot; for OnSystemRequest with url</td>
<td>Major</td>
</tr>
<tr>
<td>[GENIVI][Policy]: SDL does not apply url from PT for specified appID for OnSystemRequest</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi] INVALID_DATA received in case wayPointType with correct parameter was sent</td>
<td>Major</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>
<td>Major</td>
</tr>
<tr>
<td>GENIVI: App is disconnected due to PROTOCOL_VIOLATION when start audio streaming after rejected 2 times then accepted.</td>
<td>Major</td>
</tr>
<tr>
<td>[GENIVI]: SDL crashes during execution ATF test with start-stop SDL cases</td>
<td>Major</td>
</tr>
<tr>
<td>[GENIVI] Redundant info is sent to App when single UI got 'UNSUPPORTED_RESOURCE' from HMI</td>
<td>Major</td>
</tr>
<tr>
<td>[Genivi] SDL responses with GENERIC_ERROR instead of UNSUPPORTED_RESOURCE</td>
<td>Major</td>
</tr>
<tr>
<td>Summary</td>
<td>Priority</td>
</tr>
<tr>
<td>----------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>[GENIVI] SDL does not transfer all info parameters of unsuccess result codes when UI.IsReady = false</td>
<td>Major</td>
</tr>
<tr>
<td>[GENIVI] SDL does not respond info message in case GENERIC_ERROR watchdog timeout from HM</td>
<td>Major</td>
</tr>
<tr>
<td>[GENIVI][Navigation] SDL does not respond info message in case GENERIC_ERROR watchdog timeout from HM</td>
<td>Major</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>
<td>Major</td>
</tr>
<tr>
<td>[GENIVI] HashID should not be updated on successful single UI. if no UI.IsReady response</td>
<td>Major</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>
<td>Major</td>
</tr>
<tr>
<td>[GENIVI] SDL doesn’t set unsuccessful &quot;message&quot; value to “info” param in case HMI responds via single UI.RPC when IsReady missing</td>
<td>Major</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_prealoaded_pt.json</td>
<td>Normal</td>
</tr>
<tr>
<td>GENIVI: SDL does not send StopAudioStream() if exit app while Video service and Audio service are starting.</td>
<td>Normal</td>
</tr>
<tr>
<td>[GENIVI] The session registration is delayed by locks</td>
<td>Normal</td>
</tr>
<tr>
<td>[Genivi][SDL4.0] SDL does not create icons folder in case it was removed after SDL start</td>
<td>Normal</td>
</tr>
<tr>
<td>[Genivi][API] SDL sends BC.UpdateDeviceList with out of upper bound size of deviceList</td>
<td>Normal</td>
</tr>
<tr>
<td>[Genivi][SDL4.0] SDL sends wrong parameter names in OnSystemRequest (query_apps)</td>
<td>Normal</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>
<td>Normal</td>
</tr>
<tr>
<td>SUMMARY</td>
<td>PRIORITY</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>[SDL4.0] Genivi: Incorrect response in case Unsubscribe not supported and not yet subscribed button</td>
<td>Normal</td>
</tr>
<tr>
<td>[Genivi][Policies] Ford-specific keys are present in Genivi Policy DB - DeviceData</td>
<td>Normal</td>
</tr>
<tr>
<td>[Genivi][Policies]: &quot;application&quot; table contains &quot;certificate&quot;</td>
<td>Normal</td>
</tr>
<tr>
<td>[Genivi] SDL doesn't add device identifier to Policy DB and not present in the snapshot</td>
<td>Normal</td>
</tr>
<tr>
<td>[Genivi][Security] SDL print out CN and serialNumber details of certificates in log.</td>
<td>Normal</td>
</tr>
<tr>
<td>[Genivi][VS][AS] SDL does not send StopStream/StopAudioStream to HMI after unregistering app during streaming</td>
<td>Normal</td>
</tr>
<tr>
<td>[Genivi] Communication is not saved in SmartDeviceLinkCore.log after adding persistent data after SUSPEND</td>
<td>Normal</td>
</tr>
<tr>
<td>[Genivi] SDL doesn't transfer info parameter with UI.SetGlobalProperties to mobile.</td>
<td>Normal</td>
</tr>
<tr>
<td>[Genivi] SDL sends systemSoftwareVersion (with empty value) in RAI response if before cpu_version was invalid in GetSystemInfo.</td>
<td>Normal</td>
</tr>
<tr>
<td>[Genivi][Policies] Policy table is not initialized by SDL start without DB</td>
<td>Normal</td>
</tr>
<tr>
<td>[GENIVI] Response to UI.ScrollableMessage is sent twice to app if no HMI response HMI during</td>
<td>Normal</td>
</tr>
<tr>
<td>[Genivi] SDL doesn't send info parameter when result of ResetGlobalProperties is GENERIC_ERROR</td>
<td>Normal</td>
</tr>
<tr>
<td>Genivi: UI is waiting for TTS.IsReady timeout to elapse to send UI.</td>
<td>Normal</td>
</tr>
<tr>
<td>[Genivi] SDL response success:false to mobile app in case it received RETRY or WRONG_LANGUAGE or UNSUPPORTED_RESOURCE from HMI</td>
<td>Normal</td>
</tr>
<tr>
<td>GENIVI: SDL always responds &quot;INVALID_DATA&quot; to mobile app while receiving other code from HMI</td>
<td>Normal</td>
</tr>
<tr>
<td>[Genivi] Incorrect Version of API in MOBILE_API.xml</td>
<td>Normal</td>
</tr>
<tr>
<td>[GENIVI] SDL build without any error message with empty version in MOBILE_API.xml</td>
<td>Minor</td>
</tr>
</tbody>
</table>
SDL Core 4.3.0 Release Notes

New Features

**EXTENDED_POLICY modes:**

- The `EXTENDED_POLICY` CMake variable (previously `ENABLE_EXTENDED_POLICY`) now has three possible configurations
  - HTTP (previously `ENABLE_EXTENDED_POLICY: OFF`) - Details
  - PROPRIETARY (previously `ENABLE_EXTENDED_POLICY: ON`) - Details
  - EXTERNAL_PROPRIETARY (new, fully featured version of PROPRIETARY mode) - Details

**EXTERNAL_PROPRIETARY mode:**

**NEW POLICY TABLE UPDATE SEQUENCE**

A new policy table update flow was created specifically for the `EXTERNAL_PROPRIETARY` policy mode

- Requirements/Details
- Diagram
EXTERNAL POLICY MANAGER

As part of the \textit{EXTERNAL PROPRIETARY} policy mode, the concept of an "external policy manager" is necessary. This policy manager is a separate program which is in charge of encrypting/decrypting policy tables and attaching an HTTP header to the OnSystemRequest payload when performing a Policy Table Update.

As part of this release, a sample application which performs this function was added to this repository for those who wish to implement this new policy mode, and this program can be started along with Core using an included bash script. This sample application does nothing with the policy table snapshot during the encryption and decryption phases, allowing for OEMs to implement their own encryption algorithms in their place.

APP PERMISSIONS/USER CONSENT

Users can now control what functional groups that they want apps to be able to access, as well as decide whether to enable SDL functionality at all on a device-by-device basis.

• Logic was added to allow the user to control what devices are permitted to use SDL functionality - Details
  ◦ Users are prompted when activating an app on a new device for the first time whether or not to allow the device to use SDL functionality (sequence shown in this diagram)

• Logic was added to the Policy Manager to allow the user to control what apps have access to specific functional groups - Details
  ◦ Users are prompted when activating an app for the first time (or modifying permissions in settings) with information on what access a
requested functional group requires. The user responds to determine whether or not to allow this functionality within the new app (sequence shown in this diagram)

EXTERNAL USER CONSENT

External user consent allows the HMI to define several groups of permissions within the policy table. This allows the user to enable/disable several functional groups at once.

* The `externalConsentStatus` field is included as part of a GetListOfPermissions response from SDL Core to communicate which groups are activated - Details
* External consent groups can be used to enable sets of functional groups using the `disallowed_by_external_consent_entities_off` field in the Policy Table - Details
  ◦ If this external consent group is set to ON, all functional groupings with this parameter are allowed by the user - Details
  ◦ If this external consent group is set to OFF, all functional groupings with this parameter are disallowed by the user - Details

* External consent groups can be used to disable sets of functional groups using the `disallowed_by_external_consent_entities_on` field in the Policy Table
  ◦ If this external consent group is set to ON, all functional groupings with this parameter are disallowed by the user
If this external consent group is set to **OFF**, all functional groupings with this parameter are allowed by the user.

**CACHE MANAGER FUNCTION IMPLEMENTATIONS**

Prior to this release, several functions included in cache_manager.cc were not fully implemented and would not query the local policy table for defined rules and policies. The newly implemented functions for the **EXTERNAL_PROPRIETARY** cache manager are listed below:

- CanAppKeepContext()
- CanAppStealFocus()
- GetDefaultHMI()
- ResetUserConsent()
- GetUserPermissionsForDevice()
- GetPreconsentedGroups()
- GetConsentedGroups()
- GetUnconsentedGroups()
- RemoveAppConsentForGroup()
- GetDeviceGroupsFromPolicies()
- SetDeviceData()
- SetUserPermissionsForDevice()
- ReactOnUserDevConsentForApp()
- SetUserPermissionsForApp()
- CountUnconsentedGroups()
- SetMetaInfo()
- IsMetaInfoPresent()
- SetSystemLanguage()
- CleanupUnpairedDevices()
- SetVinValue()
Requirements/Details

**HMI_API additions:**

A new RPC was added as part of the implementation of [EXTERNAL_PROPRIETARY] policy mode

- DecryptCertificate RPC

Several API additions were made as part of the implementation of the external user consent feature

- EntityStatus enum
- ExternalConsentStatus struct
- externalConsentStatus field added to OnAppPermissionConsent and GetListOfPermissions

**Implemented proposals**

Two new evolution proposals were implemented in release 4.3.0:

- Add API Patch Version SDL-0050
- A patch version was added to the MOBILE API version, HMI API interface versions, and SyncMsgVersion struct
- External Policy Manager SDL-0045
- Details for the implementation of this proposal can be found in the External Policy Manager section of these release notes
Fixes

• Includes fixes for all defects found by the Coverity scan tool that were introduced in this release - Link
• Includes fixes for several Coverity defects that were previously implemented in the coverity branch - Link

SDL Core 4.3.1 Release Notes

Bug Fixes

• ListFiles request now queries the file system on each request instead of referring to a cached list - Link

SDL Core 4.4.0 Release Notes

Implemented Proposals

System Capabilities Query - Implementation of a new RPC which allows an app to query the capabilities of a specific component (i.e. video streaming, remote control) within a given integration of SDL Core.

Constructed Payloads - Addition of constructed payloads for control (non-RPC type) packets. These payloads allow for control packets to be more descriptive without using the overhead needed by an RPC message. The feature has been
implemented using the BSON standard through use of the bson_c_lib. This includes the introduction of protocol version 5, versions prior to this do not support this feature.

Control Frame Payloads v1.0.0 - Introduced specific parameters that will be sent with a control frame's constructed payload. This is part of the introduction of protocol version 5, versions prior to this do not support this feature.

Support Indian English and Thai - Adds the possibility to support languages English - India and Thai - Thailand.

Support For Additional Languages - Adds the possibility to support 8 new languages.

Mobile Projection - Defines new AppHMIType PROJECTION. This AppHMIType allows an app to use the same video streaming technologies as a navigation app.

Gesture Cancellation - Addition of a CANCEL element to the TouchType Enum used during an OnTouchEvent RPC.

Add Video Streaming Capabilities - Allows core to notify the proxy of the HMI's video streaming capabilities. This addition also includes a video format negotiation procedure that uses a combination of RPC and constructed payload messages.

Adding Metadata Types - Add metadataTags parameter to the Show RPC, as well as the new types MetadataTag and MetadataType. These additions allow for a more detailed description of the main field strings sent during a Show RPC request.

Human Interface Device Support - Implementation of the SendHapticData RPC. This RPC sends the HMI an array of rectangle coordinates for focusable elements used during video streaming.

Remote Control Baseline - Implementation of a new core plugin/functional module RemoteControl. This feature includes a set of RPCs that the proxy may use to control certain aspects of the HMI's climate and radio modules. Changes also include additions to the GetSystemCapability RPC and policy configurations.
Update Mobile API Mandatory Flag - Update formatting of `MOBILE_API.xml` to include the `mandatory` flag on all parameters.

Bug Fixes

Video Streaming Related

- Core still sends out deprecated Service Data ACK frames
- Proper Cleanup for the StreamerAdapter class
- SDL does not send StopStream/StopAudioStream to HMI after unregistering app during streaming

Connection Related

- TransportManager that incorrectly deletes a connection object within a C++ map iterator loop
- Core cannot find App via BT
- Connection List Lock is Not Released
- Invalid memory access in websocket_handler

Policy Related

- SDL does not set "timeout" for OnSystemRequest with url
- Policies Manager does not revert the Local Policy Table to the Preload Policy Table upon FACTORY Rese
- SDL does not write UserFriendlyMessages to DB
- SDL does not select url from PT for specified appID during GetURLs request.
- PM should verify that "seconds_between_retries" array has max length 5
- SDL does not apply url from PT for specified appID for OnSystemRequest
- PM doesn't validate the size of section "default" in "endpoints" of Policy Table
- Policy table can't be loaded when RPCs added in functional_group is greater than 50.
• PT is considered as valid with no items in "groups" sub-section from "default"

**Documentation**

• README should not reference "v4tester" application
• Create an Issue template for sdl_core

**General Fixes**

• media_manager_test fails with EXTENDED_MEDIA_MODE=ON
• Memory leak in FromMicToFileRecorderThread
• Invalid memory access in FromMicToFileRecorderThread
• SDL does not reset the default watchdog timeout of GetWayPoints request if HMI sends OnResetTimeout notification for this request
• SDL sends BC.UpdateDeviceList with out of upper bound size of deviceList
• HashID should not be updated on successful single UI if no UI.IsReady response
• OpenSDL repo still contains old "generate_test_certificates.py" script.
• SDL does not send VehicleInfo.GetVehicleData in case HMI responds invalid json
• SDL print out CN and serialNumber details of certificates in log.
• SDL returns IGNORED instead of UNSUPPORTED_RESOURCE for UnsubscribeButton
• SDL must respond with INVALID_DATA on SystemRequest that uploads a file containing "./" sequences

**SDL Core 4.4.1 Release Notes**

**Bug Fixes**

• Remote Control Applications Auto Activate Apps
SDL Core 4.5.0 Release Notes

Supported Specifications

• SDL Mobile RPC Spec: Version 4.5.0
• SDL Protocol Spec: Version 5.0.0

Implemented Proposals

Connectivity via iAP-BT and Transport Switch - Implementation of a mechanism to change a registered app connected over one transport to another seamlessly.

Mark public deprecated methods - Implemented a DEPRECATED macro for marking deprecated methods in the project. Using methods marked with this macro will result in a warning being generated.

Remove QT HMI from SDL Core (Partially Complete) - The qt_hmi component was removed from SDL Core, the QT_HMI_API interface and dbus adapter will be removed in the next major release, due to this aspect of the proposal requiring breaking changes.

Use Boost Library (Partially Complete) - The boost library is now installed as a 3rd party library, this library is currently only used in the refactored message broker component.
Enhancements

- **DBus** and **libusb** are now dynamically linked, instead of being installed as 3rd-party libraries during the SDL Core build - #2004

Bug Fixes

- Remove OEM Specific references in SDL policy table preload file
- "resultCode" should be more descriptive than "INVALID_DATA"
- SDL doesn't apply sequence SUSPEND -> OnSDLAwake -> SUSPEND -> IGN_OFF for saving resumption data
- SDL responds "resultCode: SUCCESS" while dataType:VEHICLEDATA_EXTERNTEMP is VEHICLE_DATA_NOT_AVAILABLE and not in subscribed list store
- Policies SDL should be case-insensetive to "AppID" against listed in policies manager
- App is disconnected due to PROTOCOL_VIOLATION when start audio streaming after rejected 2 times then accepted
- SDL doesn't set unsuccessful "message" value to "info" param in case HMI responds via single UI.RPC when Interface.IsReady missing
- Navigation SDL does not respond info message in case GENERIC_ERROR watchdog timeout from HMI
- SDL does not respond info message in case GENERIC_ERROR watchdog timeout from HM
- SDL doesn't send info parameter when result of ResetGlobalProperties is GENERIC_ERROR
- SDL does not send StopAudioStream() if exit app while Video service and Audio service are starting.
- 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
- IVSU SDL doesn't reject SystemRequest with filenam=IVSU but w/o binary data.
• Memory leaks: SDL does not release memory after sending AddCommand
  limit exhausted
• Negative result code send instead of IGNORED for
  UnsubscribedVehicleData when VehicleInfo IsReady Missing
• API SDL responds "UNSUPPORTED_RESOURCE", success= false in case
  only have "UNSUPPORTED_RESOURCE" to Navigation.AlertManeuver
• Default app policies are never updated after a PTU
• Build fails with GCC6+
• Remote Control test suite fails
• PoliciesManager allows all requested params in case "parameters" field is
  empty
• OnDriverDistraction SDL does not send notification to app right after this
  app changes level from NONE to any other
• Protect access to Resumption data during
  LastState::SaveStateToFileSystem
• Need to protect cache manager "pt_" from concurrent access
• Prevent deadlock in EventDispatcherImpl::raise_event
• Bluetooth StartService fail after Core restarted
• Silent error caused by implicit conversion of SmartPointer to integer
• AOA USB transport buffer size too small
• Lock screen icon URL should be updated
• Broken link in README.md for Software Architecture Documentation
• SDL doesn't send OnPermissionsChange in case of external user consent
• Build fails when ENABLE_SECURITY=OFF

Security Related Fixes

• SDL must start PTU for navi app right after app successfully registration
• SDL must start PTU for any app except navi right after app successfully
  request to start first secure service
• PolicyTableUpdate is failed by any reason and
  "ForceProtectedService"=ON at .ini file
• PolicyTableUpdate has NO "certificate" and "ForceProtectedService"=ON
  at .ini file
• SDL must respond NACK in case navigation app connected over protocol
  v2 sends StartService for audio service
• PolicyTableUpdate has NO "certificate" and "ForceProtectedService"=OFF
  at .ini file
• PolicyTableUpdate is failed by any reason and "ForceProtectedService"=OFF at .ini file
• PolicyTableUpdate is valid and brings "certificate"

ATF Software Architecture Document (SAD)

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. User Scripts Execution with ATF
    ▪ 4.3.2. User Script loading
    ▪ 4.3.3. User Function execution
    ▪ 4.3.4. Events and expectations processing
  ◦ 4.4. User Interface
  ◦ 4.5. Data View
1. Introduction

1.1. Purpose and Scope of the SAD

This document defines the high-level software architecture for the Automated Test Framework (ATF) 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>SDL</td>
<td>SmartDeviceLink</td>
</tr>
</tbody>
</table>

Definitions used in this document are in the table below.

<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>
</tbody>
</table>

For further definitions and abbreviations, please follow SDL SAD

1.3. Document Roadmap

The SW architecture of system is considered from different viewpoints:
<table>
<thead>
<tr>
<th>VIEWPOINT</th>
<th>VIEWPOINT DESCRIPTION</th>
</tr>
</thead>
<tbody>
<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>Data</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>Process State</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>Process</td>
<td>Concurrency type of view. Process View describes processes and process inter-communication mechanisms independent of physical hardware deployment.</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>
</tbody>
</table>
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:
- http://www.uml-diagrams.org/
- https://sourcemaking.com/uml

### 2. Case Background

#### 2.1. System Context, Mission and Scope

ATF is a C++/Lua framework for SDL automated black-box testing. It provides high- and low-level API for mobile applications and/or HMIs emulation.

**ATF core lua-scripts** provides API for SDL easy manipulation:
- Start and stop SDL
- Emulation websocket HMI, HMI interfaces registration and subscription
- Emulation TCP mobile connection and FORD protocol session establishing
- Expectation HMI and mobile RPC from SDL
- Expectation extensions with setting up timeouts, call-chains and fields validation
- SDL messages auto-validation

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 ATF system with enough quality and functionality that fulfill their goals</td>
</tr>
<tr>
<td>SDL Automation Test teams</td>
<td>Yes</td>
<td>Get the ATF system with enough functionality for SDL testing coverage</td>
</tr>
<tr>
<td>PM / Architect / Analyst</td>
<td>No</td>
<td>Use Customer Requirements Specification</td>
</tr>
<tr>
<td>Developers</td>
<td>No</td>
<td>Construct and deploy the system from specifications</td>
</tr>
<tr>
<td>ATF Test team</td>
<td>No</td>
<td>Test the system to ensure that it is suitable for use</td>
</tr>
</tbody>
</table>

2.3. Business Goals

ATF system allows to automatize SDL regression testing and decreasing functional end regression costs.
ATF with a smoke tests scripts suite provide Continuous testing and Continuous delivery for SDL open-source developers and integrators.

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>ATF has to be POSIX-compliant to be easily ported on all POSIX standardized OSs.</td>
</tr>
<tr>
<td>2.</td>
<td>Script language need to be used as a main-tool for simplification.</td>
</tr>
<tr>
<td>3.</td>
<td>ATF shall provide a High-level API for SDL manipulation.</td>
</tr>
</tbody>
</table>

### 3. Solution Overview

The picture below shows ATF overview.

**SEQUENCE DIAGRAM**

**SOLUTION OVERVIEW**

View Diagram
4. Views

4.1. Use Case View

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

SEQUENCE DIAGRAM

TEST DOMAIN OVERVIEW USE CASE DIAGRAM

View Diagram

SEQUENCE DIAGRAM

AUTOMATION TEST ACTIVITIES USE CASE DIAGRAM

View Diagram
Automation test activities with ATF

Test Case preparing «extend» Test execution «extend» Test result collection

Test scripts development

Test Script

SEQUENCE DIAGRAM

TEST SCRIPT EXECUTION USE CASE DIAGRAM

View Diagram
SEQUENCE DIAGRAM

SDL MANIPULATION USE CASE DIAGRAM

View Diagram
SEQUENCE DIAGRAM

MOBILE EMULATION USE CASE DIAGRAM
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.
Elements description

USER SCRIPT

• **Responsibility:**
  - Contains automated test cases written by User

• **Relations**
  - Composes *Connection Test*

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

• **Behavior**
  - *User Script* implements SDL Tests by using ATF API.

• **Constraints**
  - User scripts are not part of ATF delivering.
Application layer:

**LAUNCH**

- **Responsibility**
  - Declares a command line arguments
  - Starts *User Scripts* list one by one

- **Relations**
  - Starts *User Scripts*

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

- **Behavior**
  - *Launch* declares command line arguments and execute all left parameters as *User Scripts*

- **Constraints**
  - Needs to be run by *C++ Core*
  - Note: ATF provides *run_tests.sh* script for automation run *Launch* by *C++ Core*

**CONNECTION TEST**

- **Responsibility**
  - Provides testing API:
  - Start and Stop SDL
  - Activation and Deactivation HMI
  - Starting mobile connection and session
  - Set up HMI and Mobile RPCs expectations
  - Composes all lower level components
• Relations
  ◦ Used by User Script for SDL manipulation

• Interfaces
  ◦ Provides Test interface for User Script

• Behavior
  ◦ Connection Test works as a Facade for accumulation all ATF Functionality and providing Expectation wrappers for RPCs.

• Constraints
  ◦ N/A

Data assess layer:

CONFIG

• Responsibility
  Provides configuration for:
  ◦ setting up ATF behavior
  ◦ SDL manipulation parameters
  ◦ Emulating applications parameters

• Relations
  ◦ Used by all components
  ◦ Is filled by Utils

• Interfaces
  ◦ Provides Config interface

• Behavior
  ◦ Contains a list of defined parameters for reading and modification.
EVENTS

- **Responsibility**
  - Global events and expectations subscription with a timeout
  - Global event notification

- **Relations**
  - Used by all components.

- **Interfaces**
  - Provides *Events* interface

- **Behavior**
  - Provides centralized mechanism for components synchronization and expectation subscription

- **Constraints**
  - N/A

ATF LOGGER

- **Responsibility**
  - Logging ATF internal information
  - Providing XML reports of ATF scripts execution results

- **Relations**
  - Used by all components.
• Interfaces
  ◦ Provides **ATF logger** interface

• Behavior
  ◦ **ATF logger** provides API for logging all internal information for further SDL issues analysis

• Constraints
  ◦ N/A

**UTILS**

• Responsibility
  ◦ Support methods implementation
  ◦ Encapsulation system API calls
  ◦ Parsing command line arguments

• Relations
  ◦ Used by all components.

• Interfaces
  ◦ Provides **Utils** interface

• Behavior
  ◦ **Utils** is wrapper for **stdlib** library and **C++ Core** methods
• **Constraints**
  ◦ **N/A**

**C++ CORE**

• **Responsibility**
  ◦ Providing additional Utility object, which could not be implemented with Native Lua API or **stdlib**: timer
  ◦ Qt signal/slot system
  ◦ XML parsing
  ◦ Web-Sockets
  ◦ Tcp-Sockets

• **Relations**
  ◦ Requires Qt Framework for signal/slots system, WebSocket implementation
  ◦ Requires Lua Development Library Kit

• **Interfaces**
  ◦ Provides set of Lua interfaces

• **Behavior**
  ◦ **C++ Core** wraps Qt and Posix API for **Utils**
  ◦ Provides executable for starting Lua scripts with additional Qt meta-system functionality.
  ◦ **C++ Core** executable parses with **Launch** as a first argument, all following parameters are passed to the **Launch**.

• **Constraints**
  ◦ Requires Qt 5.3+ for WebSocket support
Requires Lua Development kit version 5.2

**Business layer:**

**SDL**

- **Responsibility**
  - Starts and stops SDL executable manipulation
  - SDL executable status monitoring
  - SDL log grabbing via telnet and save it to corresponding directory.

- **Relations**
  - Requires *Data assess layer* interfaces

- **Interfaces**
  - Provides *SDL* interface

- **Behavior**
  - Starts SDL binary and saves process id for further stop and status monitoring.

- **Constraints**
  - Requires POSIX system environment for bash script and calling OS commands like `cat`, `test`, `cat`, `sleep`

**TEST BASE**

- **Responsibility**
  - Test cases consecutive execution
  - Each test case verification check:
    - *User Script* expectation
    - SDL executable status monitoring
• **Relations**
  ◦ Requires *SDL* interface
  ◦ Requires *Events* interface

• **Interfaces**
  ◦ Provides *Test Base* interface

• **Behavior**
  ◦ *User Script* test cases are executing one by one with verifying all expectation.

• **Constraints**
  ◦ N/A

---

**VALIDATOR**

• **Responsibility**
  ◦ RPCs structure and format validation

• **Relations**
  ◦ Requires *Data assess layer* interfaces

• **Interfaces**
  ◦ Provides *Validator* interface

• **Behavior**
  ◦ Dynamically loads *HMI_API.xml* and *MOBILE_API.xml* API for further validation income RPCs from SDL.
• **Constraints**
  ◦ Requires *HMI_API.xml* and *MOBILE_API.xml* files

**Protocol layer:**

**PROTOCOL HANDLER**

• **Responsibility**
  ◦ Control and business data distributing to appropriate sessions and service
  ◦ Control messages processing
  ◦ Multi-frames assembling and disassembling

• **Relations**
  ◦ Requires *Data assess layer* interfaces

• **Interfaces**
  ◦ Provides *Protocol Handler* interface

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

• **Constraints**
  ◦ SmartDeviceLink Protocol specification

**MOBILE SESSION**

• **Responsibility**
  ◦ Sessions and services manipulation
  ◦ Sending data to mobile services
  ◦ Subscription to income mobile service data
• Heartbeat functionality

• Relations
  • Requires **Data assess layer** interfaces

• Interfaces
  • Provides **Mobile Session** interface

• Behavior
  • Accumulates API for session and services managing

• Constraints
  • **SmartDeviceLink Protocol specification**

**Transport layer:**

**HMI CONNECTION**

• Responsibility
  • Establish HMI connection to SDL
  • Sending HMI-related data
  • Subscription to income HMI data

• Relations
  • Requires **Data assess layer** interfaces

• Interfaces
  • Provides **HMI connection** interface

• Behavior
  • Wraps WebSocket connection to SDL

• Constraints
  • WebSocket API provided by Qt through **Utils**
MOBILE CONNECTION

• Responsibility
  ◦ Establish Mobile connection to SDL
  ◦ Sending Mobile related data
  ◦ Subscription to income Mobile data
  ◦ Outcome data caching to file before sending to SDL

• Relations
  ◦ Requires Data assess layer interfaces

• Interfaces
  ◦ Provides Mobile connection interface

• Behavior
  ◦ Wraps TCP connection to SDL and provides caching to file abstraction, which prevent ATF memory overflow

• Constraints
  ◦ TCP API provided by Qt throw Utils

4.3. Component Interaction View

4.3.1. User Scripts Execution with ATF

Behavior:
User starts C++ Core with a list of command line arguments. C++ Core executable launches Lua interpreter, which executes first command line argument as a script with the rest of parameters. In case of passing Launch as a first parameter Launch parse configurable parameters and consecutively load User scripts.
After Loading all User Scripts* C++ Core** execute Qt Meta-Object System, which starts Timers and Events processing.

SEQUENCE DIAGRAM

User Scripts Execution with ATF

View Diagram
4.3.2. User Script loading

Behavior:

*User Script* loads *Connection Test* as a testing and SDL manipulation API provider.

*Connection Test* loads components of *Transport*, *Protocol* and *Business* layers.

*User Script* loads *Function* from own source code.

In case of loading *Function* with a first letter in upper case *Test Base* module
interpreter it as a *Test* for further consecutive execution. All other *User Script Functions* are loaded in a common way and used as a support for *Test* functions.

**SEQUENCE DIAGRAM**

**User Script loading diagram**

**4.3.3. User Function execution**

**Behavior:**
After loading *User Script* and ATF modules, *Test Base* starts consecutive list of User *Functions* execution.
Each *Functions* has ability to manipulate SDL, HMI and Mobile connections with *Connection Test* API.
Note: Some Tests do not require all options due to emulation SDL exception cases.
For preparing consecutive test scenarios User Script able to emulate HMI, Mobile RPC and set SDL-result expectations.

SEQUENCE DIAGRAM

User Function execution diagram
4.3.4. Events and expectations processing

**Behavior:**

Events and expectation processing procedure could be spited up for:

*Setting up expectation* - *User Script* throw the *Connection Test* able to add some expectations for event and add timer for the event fail processing.

*Events processing* - On raising specific event *Events* is responsible for calling exception *Function*, on time elapse *C++ Core* directly call User or other modules *Function*.

---

**SEQUENCE DIAGRAM**

**Events and expectations processing diagram**

View Diagram
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**

View Diagram

Data flow

**HMI flow**

WebSocket raw data ➔ Json data ➔ HMI RPC

**Mobile flow**

Cached data ➔ Socket raw data ➔ Ford protocol packet ➔ Json data ➔ Mobile RPC
Elements description

GENERAL NOTE:

All following type are lua tables due to language specifics. For further information about lua tables type refer to Lua Tables Tutorial

SOCKET RAW DATA

- **Summary:**
  - Binary data stream from Mobile socket connection

- **Usage:**
  - Income and outcome data for Transport layer to OS or 3d-party library

WEBSOCKET RAW DATA

- **Summary:**
  - Binary data stream from HMI WebSocket connection
• **Usage:**
  ◦ Income and outcome data for *Transport* layer to OS or 3rd-party library

**FORD PROTOCOL PACKET**

• **Summary:**
  ◦ Protocol layer primitive with protocol related information

• **Usage:**
  ◦ Internally by *Protocol* layer for protocol header information prepossessing

**CACHED DATA**

• **Summary:**
  ◦ Data for sending to Mobile side.
  ◦ Used for avoiding flooding SDL by ATF

• **Usage:**
  ◦ Internally by *Transport* layer for Mobile-to-SDL messages

**JSON DATA**

• **Summary:**
  ◦ String data of serialized RPC in Json format

• **Usage:**
  ◦ Internally by *Business* layer as a middle state between hi-level RPC abstractions and transport/protocol layers
**HMI AND MOBILE RPC**

- **Summary:**
  - RPCs primitive types with all related fields and parameters

- **Usage:**
  - Internally by Business layer as for RPCs validation
  - By User scripts for request-response logic implementation

### 4.6. Process State View

The process State view shows the global ATF states according to system life cycle.

ATF states are related to User Script implementation and usually includes ATF waiting HMI and Mobile connections.

Following diagrams is focused on ATF Core component and shows to the User Script events processing.

**SEQUENCE DIAGRAM**

**LIFE CYCLE STATES DIAGRAM**
Elements description

INITIALIZATION

* Behaviour:
  - ATF initialize own components and load User Scripts

* Relations:
  - If all ATF subsystems and User Scripts successfully loaded, ATF starts processing Test Cases.
  - If failed, ATF is **shutting down**.

TESTING

* Behaviour:
  - ATF executes **User Scripts** test cases: emulates Mobile, HMI side, add and verifies expectations
ATF handles Events, proceeds according to own business requirements and provides to User Script verification.

* Relations:
  - ATF starts shutdown procedure on getting stop signal and in case of Critical Test Case failure.

**SHUTTING DOWN**

* Behaviour:
  - ATF stores all disconnects from SDL and deinitializes all components.

* Relations:
  - Finish ATF life cycle

4.7. Process View

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

4.8. Development View

4.8.1. Implementation Technologies

* C++11 language is selected as a programming language for ATF Core binary as a OS and CPU architecture independent.
* Qt selected as a cross-platform Framework for ATF utility support: Signal-Slot metasystem, Websocket implementation, timers and so on
* Note: Version Qt 5.3 is required for WebScoket transport support.
• Lua language selected as a lightweight, embeddable scripting language for ATF Components and *User Scripts implementation

4.8.2. Modules and Code Base Organization

Development view organizes ATF components into a separate directories:

• src - C++ source code for ATF C++ Core.
• modules - Lua components
• modules/protocol_handler - Protocol handler components
• modules/atf - Utility components

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 style
  ⌟ Compiler: GCC 4.9.3 (OS Ubuntu), Lua 5.2
  ⌟ Build system: qmake (from Qt 5.3), Cmake 2.8.12.2

• Development and testing environment for SDL Windows x64:
  • Requirements Management system: LuxProject (JIRA, Confluence)
  • Source Control System: GitHub
  • Issue Tracking System: LuxProject (JIRA)
  • Document Management System: LuxProject (JIRA, Confluence, SVN)
• C++ Coding style: Google C++ Style Guide
• Lua Coding style: Lua Style Guide

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.

**SEQUENCE DIAGRAM**

**DEPLOYMENT VIEW DIAGRAM**

ATF and SDL Core as a separate binaries could be deployed to different machines in case of using cross-machine transports (TCP,
NOTE

In common case ATF and SDL Core could be deployed on the same machine.

Elements description

ATF DIRECTORY

• Short Description:
  ◦ Contains ATF, ATF configuration files and Mobile and HMI APIs, required for SDL Core validation.
  ◦ Source code is available in https://github.com/smartdevicelink/sdl_atf

• Relations:
  ◦ Required for User Scripts execution.
• **Requirements:**
  
  ◦ System shall be compatible with Lua and Qt versions.

**USER SCRIPT DIRECTORY**

• **Short Description:**
  
  ◦ Contains **User Scripts** with Test Cases and **User common functionality** related to a different **User Scripts**
  ◦ Source code is available in [https://github.com/smartdevicelink/sdl_atf_test_scripts](https://github.com/smartdevicelink/sdl_atf_test_scripts)

• **Relations:**
  
  ◦ Requires **ATF** and **SDL**.

• **Requirements:**
  
  ◦ N/A

**SDL CORE DIRECTORY**

• **Short Description:**
  
  ◦ Contains SDL Core binaries, dependencies and configuration files

• **Relations:**
  
  ◦ Required for **User Scripts** execution.

• **Requirements:**
  
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.

ATF CONFIGURATION

ATF provides default config.lua script 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
• Path to SDL binary, HMI and Mobile interfaces
• SDL-related ATF behavior
• Reporting parameters
• List of application and they registration parameters

For further information with a list of all available parameters please refer to config.lua file.

LOGGING

ATF logging system provides following functionality:

• Logging ATF input and output data
• Storing SDL Core logs with a TCP SDL Core logger
5. View-to-View Relations

Each of the views specified in Section 4 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.
### 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

<table>
<thead>
<tr>
<th>Component / Layer</th>
<th>Application Layer</th>
<th>Business Layer</th>
<th>Protocol Layer</th>
<th>Transport Layer</th>
<th>Data Assess Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection Test</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Launch</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Validator</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Base</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDL</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mobile Session</td>
<td></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>
<td></td>
</tr>
<tr>
<td>Mobile Connection</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HMI Connection</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Config</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Events</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ATF logger</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Util</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

*Note: The table shows the mapping between elements in the Data view and the Development view.*
implemented by any selected elements from the Component view, denoted by an "X" in the corresponding cell.

<table>
<thead>
<tr>
<th>DATA / LAYER</th>
<th>APPLICATION LAYER</th>
<th>BUSINESS LAYER</th>
<th>PROTOCOL LAYER</th>
<th>TRANSPORT LAYER</th>
<th>DATA ASSESS LAYER</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMI RPC</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mobile RPC</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Json data</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>protocol packet</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Socket raw data</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>WebSocket raw data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</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. **Multi-layer architectural approach**: Transport, Business, Protocol, Application and Data Assess layers

   1. Each layer has it's own component list and provides related to layer functionality

      - *Note: In the future each layer component shall use only own or low layer interfaces*
2. Lua Script language was used due to following reasons:
   1. Lua as a lightweight and embeddable scripting language could be easily deployed to customer hardware with a limited physical resources amount.
   2. All existing script base was developed with a Lua.
   3. Lua provides a simple procedural syntax for Use Cases implementation.

3. The whole Business, Protocol, Application layers are implemented with Lua
   1. It provides an ability to dynamically extent SDL- and Protocol- related functionality
   2. It allows to cover SDL Application and Protocol layers components
   3. During User Script execution Protocol- and SDL- related functionality could easy replaced with Test-specific implementation.

4. ATF Core is on event idea.
   1. ATF provides Event system: publisher and subscriber objects, a mechanism to connect them, and event queue, containing emitted events.
   2. All internal (in ATF Core) asynchronous communication is base on the Event system
   3. All external (in User Scripts) Test Cases results waiting and delay expectation subscription is base on the Event system.

5. Qt Framework was selected due to following reasons:
   1. Signals/slots mechanism (Qt Framework Meta-Object System) for events model
   2. Cross-platform WebSocket functionality
   3. Cross-platform Timers functionality

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.

- SmartDeviceLink Protocol specification
- SDL-Core Requirements
- ATF Requirements
- Note: SDL and ATF 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_atf/issues

List of Luxoft to Ford opened question is internally available in Luxoft Jira:
- https://adc.luxoft.com/jira/issues/?jql=project in (APPLINK, SDLOOPEN, FORDUSSDL) AND issuetype = Question AND resolution = Unresolved AND labels = to_discuss AND text ~ "atf" ORDER BY key DESC
- Note: This list is handled Luxoft internally and not delivered to open-source.

List of Luxoft internal questions is available in Luxoft Jira:
- https://adc.luxoft.com/jira/issues/?jql=project in (APPLINK, SDLOOPEN, FORDUSSDL) AND issuetype = Question AND resolution = Unresolved AND labels != to_discuss AND text ~ "atf" ORDER BY key DESC
- Note: This list is handled Luxoft internally and not delivered to open-source.

6.5. Results Analysis

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

7. Lua documentation - https://www.lua.org/manual/5.3/
8. List of Figures

DEPLOYMENT VIEW

LIFE CYCLE STATES

TEST DOMAIN OVERVIEW

ATF USER SCRIPT

TEST SCRIPT EXECUTION

SDL MANIPULATION

MOBILE EMULATION

HMI EMULATION

COMPONENT VIEW
LAUNCHING USER SCRIPTS

USER SCRIPT LOADING

USER FUNCTION EXECUTION

EVENTS AND EXPECTATIONS PROCESSING

DATA FLOW

9. APPENDICES

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>0.1</td>
<td>08/16/2016</td>
<td>Initial Version</td>
<td>Elisey Zamakhov</td>
<td>SAD creation: Introduction, Case Background, Solution Overview, User Interface, Process View, References</td>
</tr>
<tr>
<td>0.2</td>
<td>08/18/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Add Use Case chapter</td>
</tr>
<tr>
<td>0.3</td>
<td>09/27/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Add Component View chapter Add Component Interaction chapter</td>
</tr>
<tr>
<td>0.4</td>
<td>09/28/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Add Data View chapter</td>
</tr>
<tr>
<td>0.5</td>
<td>10/11/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Add Development View, Process State View chapters Add Deployments View, Operational View, View-to-View Relations chapters Add Solution Background chapter</td>
</tr>
<tr>
<td>0.6</td>
<td>12/05/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Add Reference, List of Figures Update Actors in Stockholmers list, List of Figures, Overview diagram, spelling fixes</td>
</tr>
<tr>
<td>0.7</td>
<td>12/06/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>12/08/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td></td>
</tr>
<tr>
<td>0.9</td>
<td>12/09/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td></td>
</tr>
<tr>
<td>0.10</td>
<td>12/23/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td></td>
</tr>
</tbody>
</table>
### Version Data Status Author/Editor Change Description

<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>0.11</td>
<td>12/29/2016</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Update Use Cases diagrams</td>
</tr>
<tr>
<td>0.12</td>
<td>01/05/2017</td>
<td>Draft</td>
<td>Elisey Zamakhov</td>
<td>Minor spelling fixes</td>
</tr>
</tbody>
</table>

### 10.2. APPROVE HISTORY

<table>
<thead>
<tr>
<th>Version</th>
<th>Data</th>
<th>Approver</th>
<th>Approve Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>01/10/2017</td>
<td>Aleksandr Stasiuk</td>
<td>APPLINK-31326</td>
</tr>
<tr>
<td>1.0</td>
<td>00/00/2017</td>
<td>[Justin Dickow]</td>
<td>PR #YY</td>
</tr>
</tbody>
</table>