



Justin Dickow, Product Manager

Agenda

- Features
- Components
- Tools
- Resources

An enabler of features

- SDL itself is not a 'feature' for your customers
- SDL enables features in the vehicle related to brought-in device connectivity
- Every OEM doesn't have to integrate every feature

App Connectivity

- Apps can connect to the vehicle with icons displayed to the user for selection
- Apps can display text and artwork in Templates while in full screen
- Register for callbacks for spoken voice commands
- Take advantage of the vehicle's TTS and Nav engine
- Read and subscribe to vehicle data with driver and OEM permission

Owner-App

- OEMs can use SDL for owner app integration
- Independent of user interaction
- Take advantage of reading DTCs and DIDs for vehicle health reports
- Read data from the vehicle when the driver's device is connected
- More data = better future driver experiences

Video Streaming Services

- Base case is streaming moving maps
- Have seen integrations into other technologies most notably by Abalta
- SDL enabled Abalta to display their own applications on SDL implementations in vehicle (with OEM permission of course)

OEM Experiences

- A template based approach means the opportunity for highly integrated connected experiences
- No requirement for a rectangle screen displaying apps
- Freely distribute information from apps across clusters and secondary screens

Enabling the enabler

- First, permissions based architecture managed by the OEM
 - OEM decides which apps can connect, and which apps have permissions for each API
- Second, open source with a permissive license
 - github.com/smartdevicelink

SDL Components

Android

- github.com/smartdevicelink/sdl_android
- Download from source into eclipse or from jcenter for a gradle build with Android Studio

iOS

- github.com/smartdevicelink/sdl_ios
- Distributed via cocoapods and carthage
- Objective-C
- Swift supported with bridging header

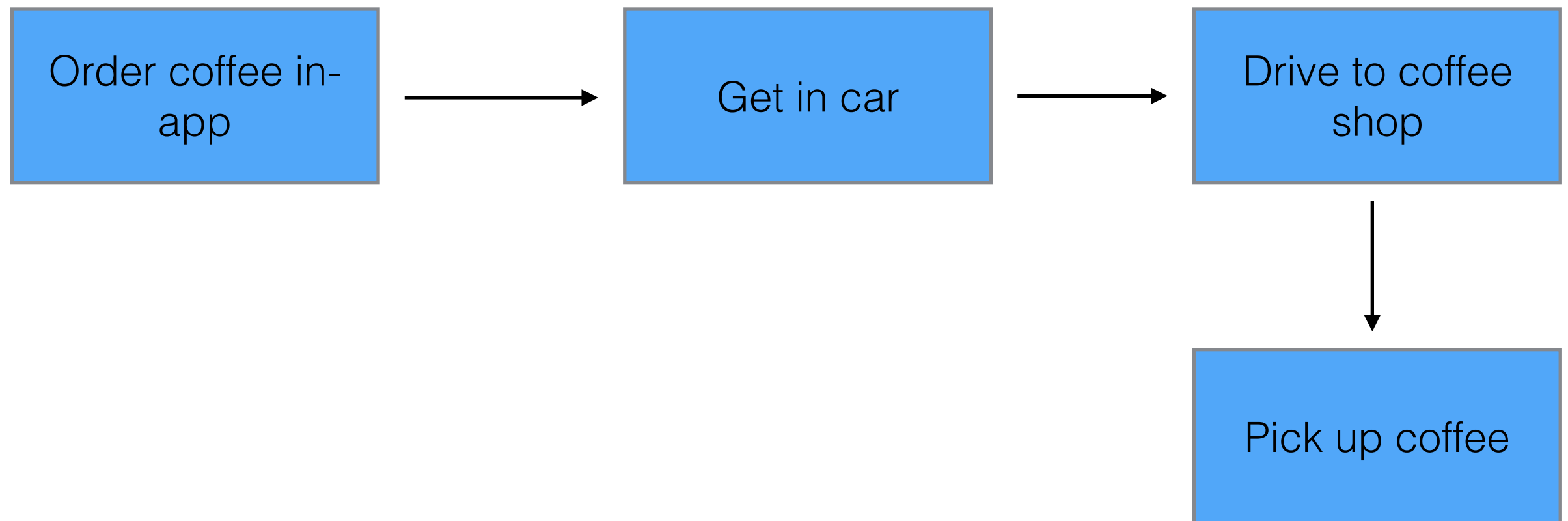
App Developer Workflow

- Defining in-vehicle use cases and user stories
- Connecting an app and experimentation

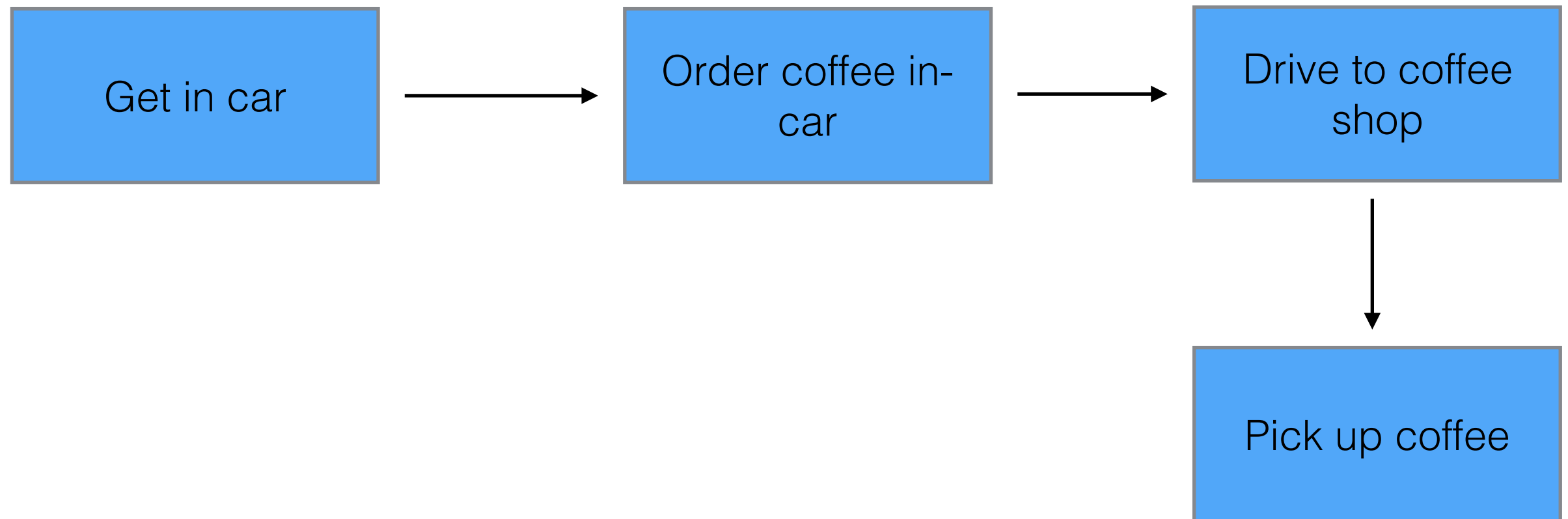
Example Use Case

- A coffee shop has an application where users purchase their coffee and can have it ready for pick up when they arrive
- How can we improve this experience?

Assumption



The experience we want



in app vs in car

1. Get phone out
2. Unlock phone
3. Find coffee app
4. Navigate to order screen
5. Confirm order

1. Phone connects automatically
2. App connects automatically
3. App recognizes you're in the car at a time when you normally get coffee
4. Alert asks if you'd like to place your usual order
5. You say yes

Experimenting with SDL

- This user experience can go beyond the simple user flow we've described
- Relevant APIs
 - SendLocation to navigate user to coffee shop
 - Alert to ask user if they'd like to order
 - SoftButtons to allow user to choose from favorites

Core

- github.com/smartdevicelink/sdl_core
- This is the component embedded in the vehicle
- It is the middleware between your HMI and the connected application

Core Configuration

- https://github.com/smartdevicelink/sdl_core/blob/master/CMakeLists.txt
- EXTENDED_MEDIA_MODE: support for video streaming capabilities
- BUILD_BT_SUPPORT: include the default bluetooth transport
- BUILD_USB_SUPPORT: include the default AOA transport
- ENABLE_LOG: view logs in console
- BUILD_TESTS: enable unit tests with `make test`

Other Configurable Options

- https://github.com/smartdevicelink/sdl_core/blob/master/src/appMain/smartDeviceLink.ini
- A host of runtime configurations for SDL
- Enable/Disable policies
- Video Streaming configuration
- Request rate limiting and timeout parameters

Transport Adapters

- The first major responsibility of an OEM integrating SDL Core
- The abstraction between your vehicle's hardware transport (USB, BT, iAP) and SDL Core
- SDL ships with BT and USB transports but your drivers may be different on your hardware
- <https://smartdevicelink.com/guides/core/transport-manager-programming/>

SDL HMI

- The second major OEM responsibility
- This is how the driver interacts with SDL applications connected to your vehicle
- SDL uses templates to display information coming from applications
- The HMI implementing a request/response/notification communication with SDL over WebSocket to send and receive information about the current state of the application and to notify SDL of user interaction with the HMI in the vehicle

Connecting with WebSocket

```
let url = "ws://localhost:8087"  
  
var socket = new WebSocket(url)  
  
socket.onopen = function (evt) {  
    registerComponents(socket)  
}
```

Registering Components

```
registerComponents() {  
    var JSONMessage = {  
        "jsonrpc": "2.0",  
        "id": -1,  
        "method": "MB.registerComponent",  
        "params": {  
            "componentName": "UI"  
        }  
    }  
    ...  
}
```


Simple Messaging Format

```
// SDL takes care of registration of apps entirely, all you need to  
know is that there are apps (and handle RPCs in general)
```

```
handleRpc(rpc) {  
  
    let methodName = rpc.method.split(".")[1]  
  
    switch(methodName) {  
  
        case "UpdateAppList":  
  
            updateAppList(rpc.params.applications)  
  
    }  
  
}
```

Communicate User Actions to SDL

```
// user selects an app we just send..
```

```
static OnAppActivatedNotification(appID) {  
  
    return ({  
  
        "jsonrpc": "2.0",  
  
        "method": "BasicCommunication.OnAppActivated",  
  
        "params": {"appID": appID}  
  
    })  
  
}
```

Templates

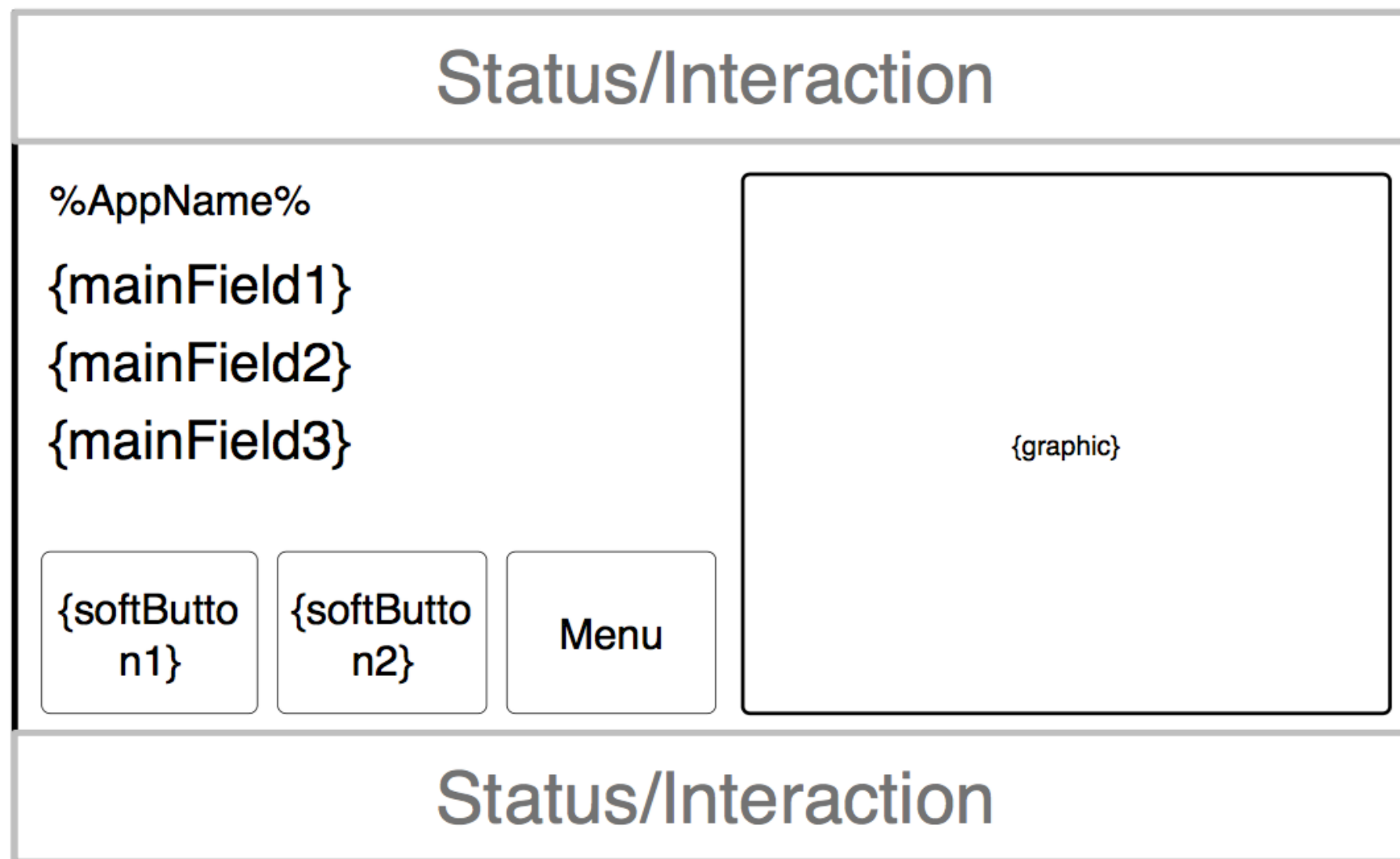
- Suggest to OEMs to support a variety of templates under the predefinedLayouts enum set by the app using SetDisplayLayout
- Templates include text fields, graphics, soft buttons, and subscribe able buttons
- Templates enable additional use cases for app developers

Use Case - Contacts App

- An application would like to allow a user to
- Search for favorite contacts
- Display info about a contact
- Call the contact

Template 1

TEXT_AND_SOFTBUTTONS_WITH_GRAPHIC

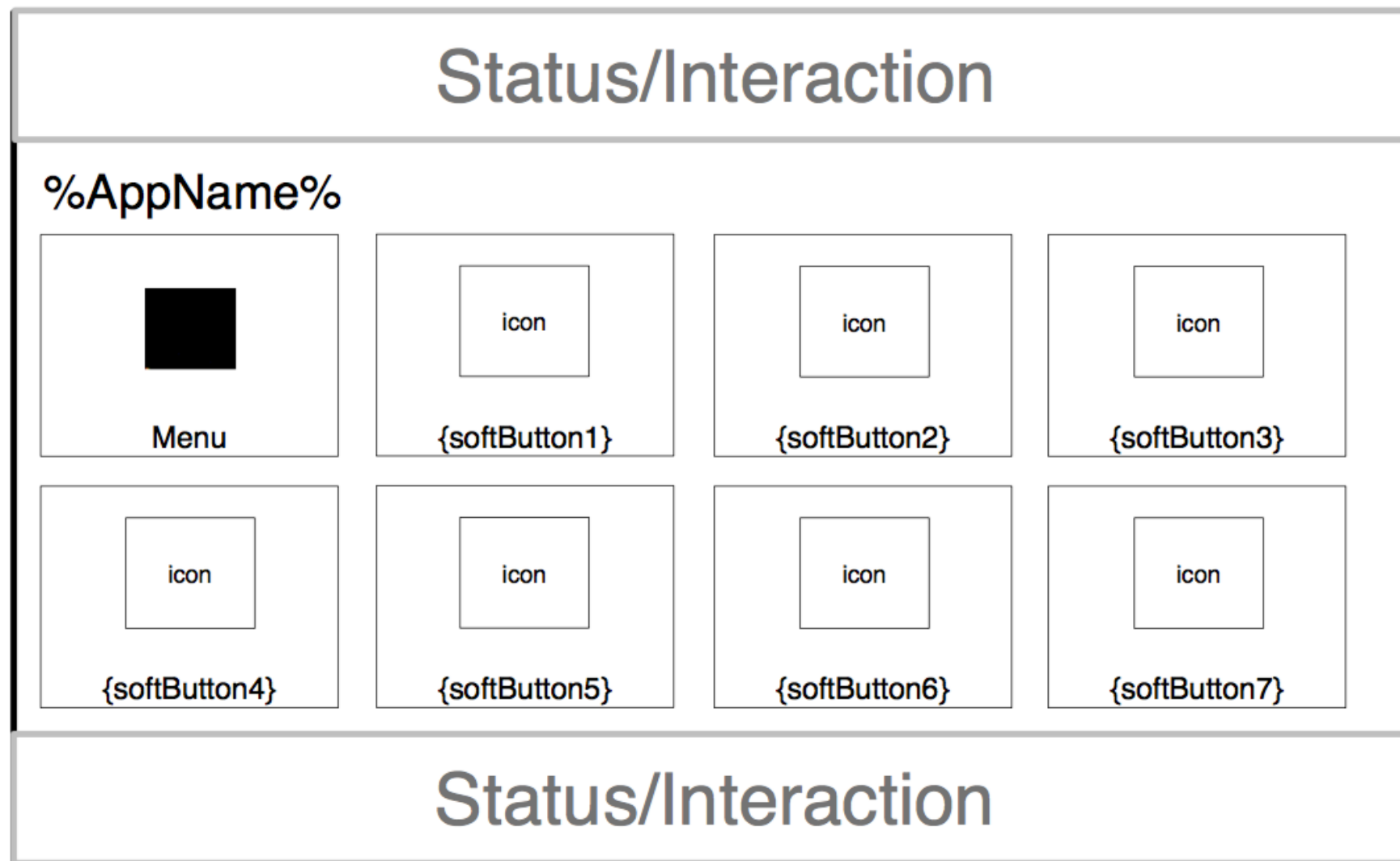


Displaying the contacts

- Some apps might use the menu and `addCommand` so the user can press menu to view contacts in a list view
- Another way is to use another template!

Template 2

TILES_ONLY



Templates don't have to be static

- Consider: a media application being used
- Corner Case: Artwork is not available
- Bad: Have an empty square where an image should be
- Option: Dynamic templates - accommodate what's available

MENU

Media Player



St. Vincent
Every Tear Disappears
St. Vincent

1:36 / 4:09



MENU

Media Player

St. Vincent

Every Tear Disappears

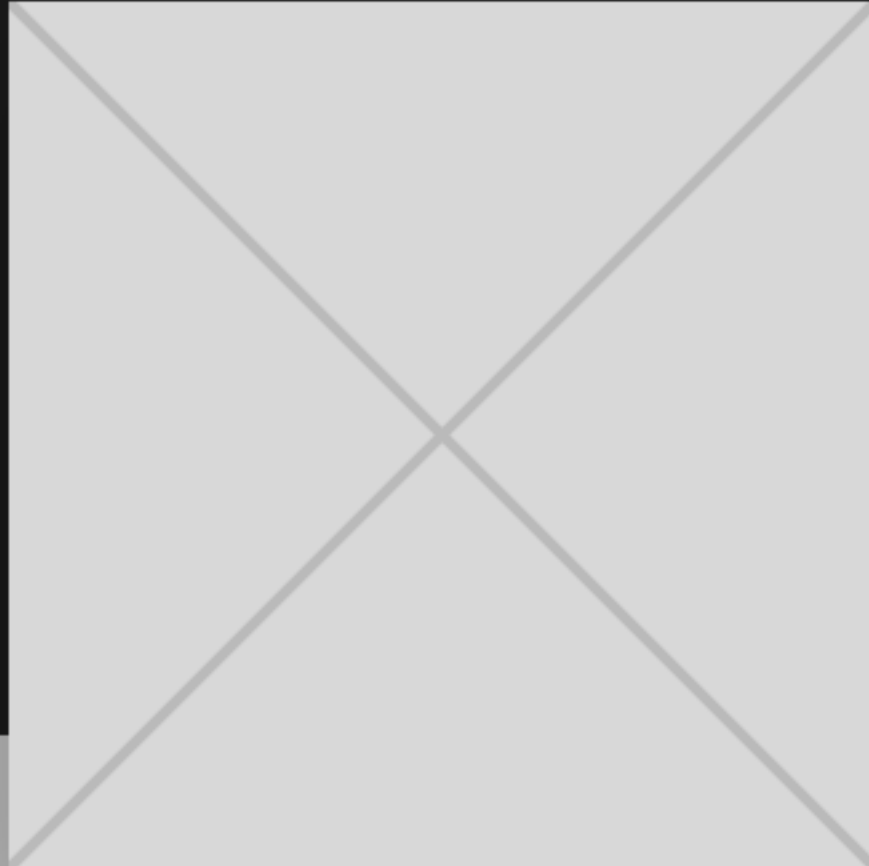
St. Vincent

1:36 / 4:09



MENU

{mainField4}

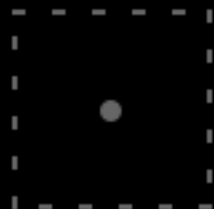


{mainField3}

{mainField1}

{mainField2}

{currentTime} / {endTime}



Server

- Enable cloud management of Policies
- Third major OEM responsibility
- Simple JSON format, optional encryption
- Livio Reference Implementation
(policies.smartdevicelink.com)

How Policies Work

- Applications receive an AppId
- OEM Server is configured to associate the AppId with RPC and data permissions
- Head Unit triggers a policy table update for unknown AppIds, after x ignition cycles, etc.
- Every RPC is checked against the latest policy table for permission
- AppIds can be revoked and reissued
- You ship SDL with a preloaded policy table `build/src/appMain/sdl_preloaded_pt.json` which is loaded on first run

Policy Table Sections

- Module Config - Runtime config for policies
- Functional Grouping - Related Groups of RPCs to be referenced in permissions
- Consumer Friendly Messages - Text strings localized for SDL
- App Policies - Permissions definitions for applications

Policies Example

```
"app_policies": { // The section
    "default": { // Default permissions for all apps
        "keep_context": false, // Apps cannot persist alerts
        "steal_focus": false, // Apps cannot force full screen
        "priority": "NONE", // Apps have no priority
        "default_hmi": "NONE", // Apps default to no HMI status
        "groups": [ // union of groups create policies
            "Base-4" // The group of RPCs allowed by default
        ]
    }
}
```

Policies Example Cont.

```
"Base-4": { // The Base-4 functional group

  "rpcs": { // Base-4 consists of the following rpcs

    "AddCommand": { // Apps with Base-4 permission can use AddCommand

      "hmi_levels": [ // In the following HMI states

        "BACKGROUND",

        "FULL",

        "LIMITED"

      ]

    },

    "AddSubMenu": { // Apps with Base-4 permission can use AddSubMenu

      "hmi_levels": [ // In the following HMI states

        "BACKGROUND",

        "FULL",

        "LIMITED"
```


Tools

- RPC Builder - iOS app to build messages manually to send to core, great for testing
- sdl_hmi - sample SDL hmi that we use for testing but you can use for experimenting as well
- Relay App iOS

The problem

- Getting logs in Xcode console requires a usb connection to the debugger
- Connecting to AppLink with an iOS app requires a usb connection to SYNC
- Therefore - you can't log and connect at the same time

Relay App (Solution)

- https://github.com/smartdevicelink/relay_app_ios
- Connect relay app to SYNC over usb
- Connect your app to Xcode over usb
- Connect relay app to SDL via tcp with provided IP address on relay app screen

Resources


- There's a lot of information, SDL is a whole ecosystem
- This document is a good overview but..

smartdevicelink.com


Global Search

sdl


🔍 BasicCommunication.OnReady|

 **BasicCommunication**
Found in : Documentation / HMI




 **BasicCommunication**
Found in : Documentation / HMI



 **BasicCommunication**
Found in : Documentation / HMI



 **OnReady**
Found in : Documentation / HMI



API Reference

Documentation HMI

[Overview](#)
[WebSocket Transport](#)
[Getting Started](#)
[BasicCommunication](#)
[UpdateDeviceList](#)
[ActivateApp](#)
[MixingAudioSupported](#)
[AllowDeviceToConnect](#)
[SystemRequest](#)
[GetSystemInfo](#)
[OnReady](#)
[OnStartDeviceDiscovery](#)
[OnDeviceChosen](#)
[DialNumber](#)
[OnAppActivated](#)
[OnAppDeactivated](#)
[OnAppRegistered](#)
[OnAppUnregistered](#)

OnReady

- Type: Notification
- Sender: HMI
- Purpose: Inform SDL about readiness to communicate

OnReady is the first message which begins the SDL-HMI communication after the WebSocket transports are established.



MUST

The HMI must send this notification after the connection is established and the HMI is ready for communication in order to communicate with SDL.

Example Notification

```
{
  "jsonrpc" : "2.0",
  "method" : "BasicCommunication.OnReady"
}
```

Guides and Tutorials


Guide iOS

←	Guide
Filter...	
Getting Started	
Software Architecture	
Mobile Navigation	

Mobile Navigation Technical Introduction

V 0.1.0

SmartDeviceLink (SDL) enables smartphone navigation applications to stream video and audio to the head unit. This enables a brought in navigation experience that is on par with the embedded navigation.

 **NOTE**
In order to use SDL's Mobile Navigation feature, the app must have a minimum requirement of iOS 8.0. This is due to using iOS's provided video encoder.

SDL Basics

The mobile navigation feature uses the standard SDL library, but has a different behavior on the head unit. The main differences are:

Open Source Documentation!

[View on GitHub.com](#)



[Previous Section](#)

[Next Section](#)



https://github.com/smartdevicelink/sdl_hmi_integration_guidelines

https://github.com/smartdevicelink/sdl_ios_guides

https://github.com/smartdevicelink/sdl_core_guides

SDL Slack Channel



+



Join **SmartDeviceLink** on Slack.

20 users online now of **160** registered.

GET MY INVITE

or [sign in](#).

Thank you