sdl

Best Practice Guides

Document current as of 03/12/2019 04:04 PM.

Display Information

Display vs Voice

Designing your application on a mobile device is entirely a visual process. However, designing application for in-vehicle experience requires closer consideration of driver distraction rules to ensure the safety of your application user. Below listed are some of the best practices you will need to consider while designing your application for in-vehicle use so as to eliminate the need to understand the driver distraction laws and guidelines that are continually changing around the world.

General Guidelines

USE VOICE COMMANDS

It is highly recommended to focus on voice interactions first and foremost. In a vehicle, especially while driving, your application usage should be almost entirely accomplished using voice commands.

PRESENTING CLEAR INFORMATION

There are only a few ways of presenting information to the user. The first way is to write to the available *templates*, for example to your application's main screen. While more advanced displays allow for more updated fields and *templates*, you are always given a *base template* to write to.
The second is to send a text string for the voice engine to process and speak back to the user. The combination of the two, display *template* updates and sending text to be processed into speech, is the best way of presenting information to the user. An example of this is requesting a pop-up with text-to-speech (*TTS*), or `Alert`.

TBD: Attach link for Available Templates.

USING DISPLAY

There are anywhere between two and four lines of text available to your application via the `Show` command. As a general rule, use this display to convey the current state of your application.

line 1 and *line 2*: Use *line 1* and *line 2* for relatively continuous updates such as, what station is playing, artist/track being streamed, distance to location, etc.

line 3 and *line 4*: Other information which might be useful, but not pertinent to functionality are best included on *Lines 3* and *4*, if available.

NUMBER OF CUSTOM SOFTBUTTONS

When using `Show` in combination with `SetDisplayLayout`, depending on the OEM, the number of custom softbuttons on each display layout will differ. The standard format of softbuttons for each system is as follows:

LAYOUT	STANDARD # OF SOFTBUTTONS
DOUBLE_GRAPHIC_WITH_SOFTBUTTONS	8
GRAPHIC_WITH_TEXT	0
GRAPHIC_WITH_TEXT_AND_SOFTBUTTONS	2
GRAPHIC_WITH_TEXTBUTTONS	3
GRAPHIC_WITH_TILES	3
LARGE_GRAPHIC_ONLY	0
LARGE_GRAPHIC_WITH_SOFTBUTTONS	8
MEDIA	8
NON_MEDIA	8
TEXT_AND_SOFTBUTTONS_WITH_GRAPHIC	2
TEXT_WITH_GRAPHIC	0
TEXTBUTTONS_ONLY	8
TEXTBUTTONS_WITH_GRAPHIC	3
TILES_ONLY	7
TILES_WITH_GRAPHIC	3

PROVIDE FEEDBACK

- Brand new user
  A welcome message with basic instructions could be given the first few times a user uses the application in vehicle using the `Speak` function. Once they've used the application a set number of times, you can remove these helpful prompts.
  Send a `Show` command to update the display with a welcome message or initialization message such as "*Buffering...*".
- Logged In User Account
  If your application does require a logged in user account, you should always have logic to catch applications that are running in-vehicle without an active account and display a message notifying the user to log in when not driving.
  Refer to the Standard Interaction Phrases section for sample voice and/or text messages.
- Waiting for Online Data Download
  If your application is waiting for online data to get downloaded, display "*Loading...*" text on *Line 1* of the *template* to inform the user of the download.
- Background Application
  If your application is a background application, with no real action for the user to take while using it in-vehicle, you might not have any voice commands or soft-buttons available in your application. It is recommended to provide an "*About*" or "*Info*" Menu/voice command or/ and soft-button (with voice command) which when used by the user provides a brief message about the application. You may also choose to point them to the help section in your mobile app which might be more descriptive.
- Dealing with Permissions
  If your application is using certain functions, such as vehicle data, it is

essential to ensure that the application provides feedback to the user in cases when:

    - All or part of the vehicle data is not available in the vehicle (as some car lines or model only provide a subset of the vehicle data available in other car lines).
    - User did not provide consent to use vehicle data.
    - User has disabled access to all or part of the vehicle data your application is using.

  For the above scenarios, as a feedback you may let the user know that the application might not work as expected due to the lack of availability of vehicle data. Refer to the Standard Interaction Phrases section for sample voice/text messages.

  Note that some RPCs are protected by policies for every OEM. To gain access to such RPCs, contact the OEM.
- Using ScrollableMessage
  `ScrollableMessage` is a useful way to show and display text to a user. In some markets like Europe and Asia, `ScrollableMessage` can be used at any time, to show a message or other relevant information that wouldn't fit on an `Alert`. It is important to note that in North America, `ScrollableMessage` is limited in use and cannot be used while the vehicle is in motion. Please be advised that you may find these messages are blocked from being shown in that region. To detect this condition, use the `OnDriverDistraction` notification on vehicles before showing the message, or if you detect `ScrollableMessage` is rejected due to driver distraction restrictions, simply use a `Speak` or another
  method to convey the information.
- Handling API rejections Part of guides
  Depending on the state of the system, or your application permissions, some of the messages you send to the vehicle may get rejected. It is important to understand why these are happening when they occur, so that you may present the proper information to the user. If a command you send is `REJECTED`, it is recommended to send it again, depending on the message. For example, if an `AddCommand` is rejected, you will need to send it again, to ensure your menu option or voice command is available on the system. If a request for `Alert` is `REJECTED`, it may mean the system is in a state in which your message cannot be played.

You may want to try again after 15-30 seconds if the information is pertinent to the user.

- General feedback recommendation
  - Applications must provide an audio or visual response to user interactions (button presses, *VR*, `AddCommands`, etc.).
  - Applications must not provide an incorrect or unexpected response to user interaction.
- Standard Interaction Phrases
  - Login Required
  - Text: To use < *app name* > you have to be logged in.
  - Voice: To continue using < *app name* > please login on your mobile phone while not driving.
  - Vehicle Data Availability
  - Text: Grant access to vehicle data in Mobile Apps settings.
  - Voice: < *App name* > might not work as expected as we are unable to access < *Vehicle Information, Push Notifications, Location Information and Driving Characteristics* >. Please enable this feature in Mobile App's settings menu while not driving.

Graphics

When implementing graphics into your apps you should make sure that the app looks good in both day and night mode. For graphics this implies that you should at either have a background or work with outlines. For icons, outlines are needed to increase contrast in all situations. Light icons should have a dark contrast and dark icons should have a light contrast.

Please ensure to use the correct dimensions for your images.

Graphics used in `choiceSets` must be uploaded before the `choiceSet` is used. This will decrease performance if many different graphics are to be used in one `choiceSet`. To mitigate that the usage of generic icons is encouraged (like a CD icon instead of the actual Album art).

Add Links to uploading graphics (android/ iOS) sections

Driver Distraction Rules for Graphics

Due to driver distraction rules the graphics used can’t incorporate text or any form of data or graph. No moving images or video can be used, or any graphic that a user could interact with. If your app incorporates social media you are not allowed to display any graphics from social media posts or any form of attachment.

Languages

General Guidelines

As a developer, you will want to localize your application, via voice and on the display to match the experience with the rest of the user interface inside of the vehicle.

LOCALIZATION AND HANDLING LANGUAGE CHANGES

In order to properly handle language changes inside of the vehicle, we propose the following rules.
1. Register your application with the language it currently supports on the phone's user interface.

2. Check the display and voice language that the vehicle is currently set to in the `RegisterAppInterface` response.
3. If your application does not support the language in the vehicle, you may send commands as usual. By default if the user starts your application in the vehicle, `the vehicle will notify the customer the languages between the vehicle and phone does not match` and no other steps are required.
4. If your application supports the language(s) in the response value, send a `ChangeRegistration` message with the vehicle language and proceed to adding your commands and bootstrapping your application.

DYNAMIC SWITCHING OF LANGUAGES:

The best practice for your SDL integration is a dynamic switching of the Apps SDL language. The `RegisterAppInterface` RPC will report the head unit's language. If this language differs from the current language your App is set to, you should dynamically reload the strings in the language to that set in the head unit. The driver is used to the language in his car, therefore your app should be aligned with the language in the vehicle.

Note: If dynamic switching of languages is not a possibility for your application (because you load strings dynamically from your backend based on the user profile language settings), you may still choose to keep your language. `The user will be informed from the head unit that the App's language differs from SYNC and that Voice commands will not work as expected.` Check out Regional Language Switching section for best practices to minimize false notifications to the user.

APP'S TTSNAME

It is always important to keep in mind the voice engine in the head unit will always pronounce the given data according to the language it is set to.

Example:
App Name "Livio Music" sounds like "Leeveeo Moozik" (which is not how the app name
is supposed to be pronounced) if the head unit is set to German.

To prevent your app name from being pronounced in from happening and for the user to be able to correctly activate your app via voice as well as for the head unit to correctly pronounce the App Name, you will have to use the two parameters `TTSName` and `VRSynonyms` in the `RegisterAppInterface`. Both fields have to be filled with a `String` that represents the pronunciation of your App name in the current language.

Example:
Fill `TTSName`and one value of `VRSynonyms` with `Leeveeo Moozik` when the
Livio Music connects to a German head unit to get the correct pronunciation
of the app name in German. This is the only way to enable the user to start
the app with the english pronunciation of Livio Music and also allow the head
unit to pronounce name of the app correctly.

Note: The recommended best practice mentioned above must be followed only if the app name sounds wrong in the language the app switches to.

REGIONAL LANGUAGE SWITCHING

If your app only supports one variant of English:
You should update the registration information using the `changeRegistration` RPC (you don't have to re-register, because you don't have to change the `TTSN`

ame). As soon as you get the information that the user's vehicle is set to a different regional version of the language, you should call the ChangeRegistration RPC with the actual regional variant that the user's vehicle is set to. There is no other change required.

```
Example:
App is set to "EN_US".
RegisterAppInterface reports the head unit is set to "EN_GB".
Use "ChangeRegistration(EN_GB, EN_GB)" to update the language.
Note: You don't have to update either "TTSName" or "VRSynonyms".
```

PERSISTED LANGUAGES

If your App's languages differs from your SDL languages, the App should store this different language and use it the next time it registers with the head unit.

The above recommendation also covers the use case where the user has his vehicle always set to a different language than the language on his mobile phone. If the app did not persist the language it would have to switch every time, which degrades the apps performance.

Menu Items and Interactions

All the major features or functions of your application (nouns/verbs) could be loaded as top-level voice commands and or menu items using AddCommands .

It is recommended that this list be less than 150 commands, to improve application initialization performance and for high voice recognition quality.

General Guidelines

RESPONDING TO USER INPUT

As an application, you will want to respond to user input. To limit changes in modality, we highly recommend looking at the trigger source for input provided to your application, either via voice or menu. In instances where `OnCommand` indicates you've made a selection, and you want to use `PerformInteraction`, ensure the mode is the ***same***. For example, if trigger source is `MENU`, you'll want to start your interaction with an interaction mode of `MANUAL_ONLY`. If trigger source is `VR`, you'll want to trigger your interaction as `BOTH` or `VOICE_ONLY`. Also, as a best practice always set your `PerformInteraction` timeout to max giving sufficient time for the user to respond.

HANDLING CHOICES, CHOICESETS AND COMMANDS

Applications can load `ChoiceSet` with 100 items. If your `PerformInteraction` requires additional commands, you may reference additional `ChoiceSets`. That is, one `PerformInteraction` can contain more than one `ChoiceSet`. If your list of choices is known ahead of time, it is helpful to create these during your initialization phases, and simply reuse the `ChoiceSet` throughout your application's lifecycle. When a user initiates an interaction, the user may choose from whatever choices are defined at that moment. It is up to the application to ensure that all appropriate choices are defined ***before*** the app interaction. Also, consider grouping your choices in a way that maximises reusability of the defined `Choices` or `ChoiceSets`.

Note:
It is not recommended to consistently delete and create choice sets. If you must delete a `ChoiceSet`, it is suggested that you wait some time since it was last used. Immediately deleting a `ChoiceSet` after its `PerformInteraction` has

returned could lead to undesired application behavior.
While `DeleteCommand` and `DeleteInteractionChoiceSet` are supported RPCs, only use them when appropriate. Avoid deleting commands and `ChoiceSets` that will knowingly be used again.
* Every `choiceID` across different `ChoiceSets` should have unique IDs within the app's lifecycle.

ORDERED ITEMS

Always order items in your lists for better user experience. Ordering the items which are more oftenly used in your application or ordering the items based on the specific user preferences, or based on current location of the user etc. will help keep the most important items to the user accessible. In general, order the items from most important to least important.

VR FOR SOFTBUTTONS

To add VR for softbuttons, use `AddCommand` with no `menuParameters`.

NAME AND IMAGE

You may use both name and image for items wherever applicable.

CHOOSING VOICE GRAMMAR

Applications must not register voice grammars using synonyms that include other application names, or conflict with on-board voice commands.

MAKING LISTS MORE INFORMATIVE

It is always a good practice to add relavent and useful information to the list items, for example if your app is a media app and you have a list of audio contents, adding information such as when the content was aired.

HANDLING LISTS

PerformInteraction lists should always be as small as possible by UX design. However, in cases where having long lists cannot be avoided, please follow the below best practices for better user experience.

Note: Avoid opening `PerformInteraction` from within another `PerformInteracti on` .

1. List with multiple action item per choice
    - For each item which has several more actions available:
      Example: Searching for events in the vicinity. There might be a big number of events. On each event you can select “Call venue”, “Navigate to it”, “Details”, “Play” (playing the artist of a potential concert in the vicinity).
        - Present each result on it's own screen via `Show` and announce it with a pure voice alert via `Speak`.
    - Define softbuttons for each possible action and add the respective command to voice and menu.
    - To cycle through the result list, you may use one of the below options:
    - `Skip` button (which is currently only available for Media Apps)
    - Softbuttons & Voice commands
    - Show the current item and the length of the list in the Media-track. For example 1/10 for the first item out of 10 items from the result.

- Announce the availablility of `Skip`hard button on the steering wheel for easy navigation.

Note: Only make the announcement the first time the search has been completed to reduce any kind of annoyance to the user.

- For each item only one action available:
  Example: scenario is already "Navigate To _"
  In contrast to the above, you may choose to use `PerformInteraction`.

2. Simple lists
   If the results are known by the user or the result list is very small you can use `PerformInteraction` instead depending on the current use case.

Example:

```
User Action 1:

VR/MENU: "Create result list"
Result 1:

  SCREEN LINE 1:  "Result 1 a"
  SCREEN LINE 2:  "Result 1 b"
  MEDIA TRACK:    "1/25"
  SOFTBUTTONS:   Action 1, Action 2, Action 3,...,Previous, Next, More...
  TTS:           "Result 1 <possibly more information about the result>"
                //First Time: Announce possible ways to navigate (skip
buttons,
                  voice command, etc..)
```

User Action 2:

VR/MENU/SOFTBUTTON: "Action 1"
Result 2:

This is only applicable if the "Action 1" of "Result 1" results in another screen. If just a call is initiated by "action 1" there is no need to display an ensuing screen.

SCREEN LINE 1: "Action 1 Result 1 a"
SCREEN LINE 2: "Action 1 Result 1 b"

SOFTBUTTONS: Softbuttons highly depend on the use case you are developing.
If the user again has an array of choices, use the same concept as explained above. Otherwise, adapt the screen to your use case.
TTS: "Result of Action 1 on Result 1"

User Action 3:

VR/MENU/SOFTBUTTONS: "Next"
Result 3:

SCREEN LINE 1: "Result 2 a"
SCREEN LINE 2: "Result 2 b"
MEDIA TRACK: "2/25"
SOFTBUTTONSS: Action 1, Action 2, Action 3,...,Previous, Next, More…
TTS: “Result 2 <possibly more information about the result>.”

PerformAudioPassThru

The `PerformAudioPassThru` RPC feeds you audio data from the vehicle’s microphone. The audio data can be used in cloud-based and on-line voice recognition to achieve dynamic user interaction, such as POI (point of interest) search, information query, or even record when the driver is singing. The audio data will be in uncompressed PCM format. The sampling rate, bit width, and timeout can be set, however, the supported parameters will be sent in the `registerAppInterface` response. Generally, 16 bit width, 16kHz sample rate will be supported.

The parameter `muteAudio` is used to define whether or not to mute current audio source during AudioPassThru session.

When the `PerformAudioPassThru` is used for voice recognition, `muteAudio` should be set to true to minimize audio interference.

If you want to mix the input audio from `PerformAudioPassThru` session with current audio source, eg. a karaoke app recording both the user's voice and the background music, you can set `muteAudio` to false.

`onAudioPassThru` keeps you updated with the audio data transfer every 250ms.

`EndAudioPassThru` enables you to end the audio capture prematurely. This is useful if your app analyzes the audio level and detects that the user has stopped speaking.

Additional notes about the audio data format:

- There is no header (such as a RIFF header).
- The audio sample is in linear PCM format.
- The audio data includes only one channel (i.e. monaural).
- For an 8 bit width, the audio data is unsigned. For a 16 bit width, it will be signed and little endian.

Audiobooks

An audiobook app for SDL should include as much of the native apps functionality as possible, since that is what a user will be expecting out of the app.

USE VOICE COMMANDS

Your Audiobook app should contain such basic functionality as `Play`, `Pause`, and `Resume`. You can use build in controls for these features by calling the `subscribeButton` RPC. This will allow you to not only create buttons on the infotainment system but also be informed when the user presses hard buttons, which might be placed on the steering wheel. It is important to also add voice control capabilities for these functions. To add a voice command use the `addCommand` RPC but omit the `menuParams` argument. This will create a voice command without creating an item in the menu.

Add link to AddCommand Section

DISPLAY INFORMATION

Your app should display information like title, author, and potentially additional items like chapter number. It is encouraged to use the following layout of information on the screen:

TEXTFIELD	INFORMATION
mainField1	Title
mainField2	Author
mainField3	Chapter (x/x)

It is encouraged to show static images of the book cover, however moving or interactive images are not allowed. Neither is the display of the actual text of the book.

Add link to DisplayInformation Section

MANAGING LISTS

Use the `performInteraction` RPC to allow users to choose from lists of Books.

MANAGING AUDIO

When you receive the `NOT_AUDIBLE` state you should pause the audio and resume when you receive `AUDIBLE`. The pausing and resuming should be independent on the current `HMILevel` state.

Music Apps

When creating a music app for SDL the app should include as much of the native apps functionality as possible. If your app requires a user name and password to be used then the user should be informed for this instead of the user just being unable to access the apps functionality. Please refer to the `Displaying Information` section.

BASIC COMMANDS

Your Music app should contain such basic functionality as `Play`, `Pause`, and `Resume`. You can use build in controls for these features by calling the `subscribeButton` RPC. This will allow you to not only create buttons on the infotainment system but also be informed when the user presses hard buttons, which might be placed on the steering wheel. It is important to also add voice control capabilities for these functions. To add a voice command use the `addCommand` RPC but omit the `menuParams` argument. This will create a voice command without creating an item in the menu.

Add link to AddCommand section

DISPLAY INFORMATION

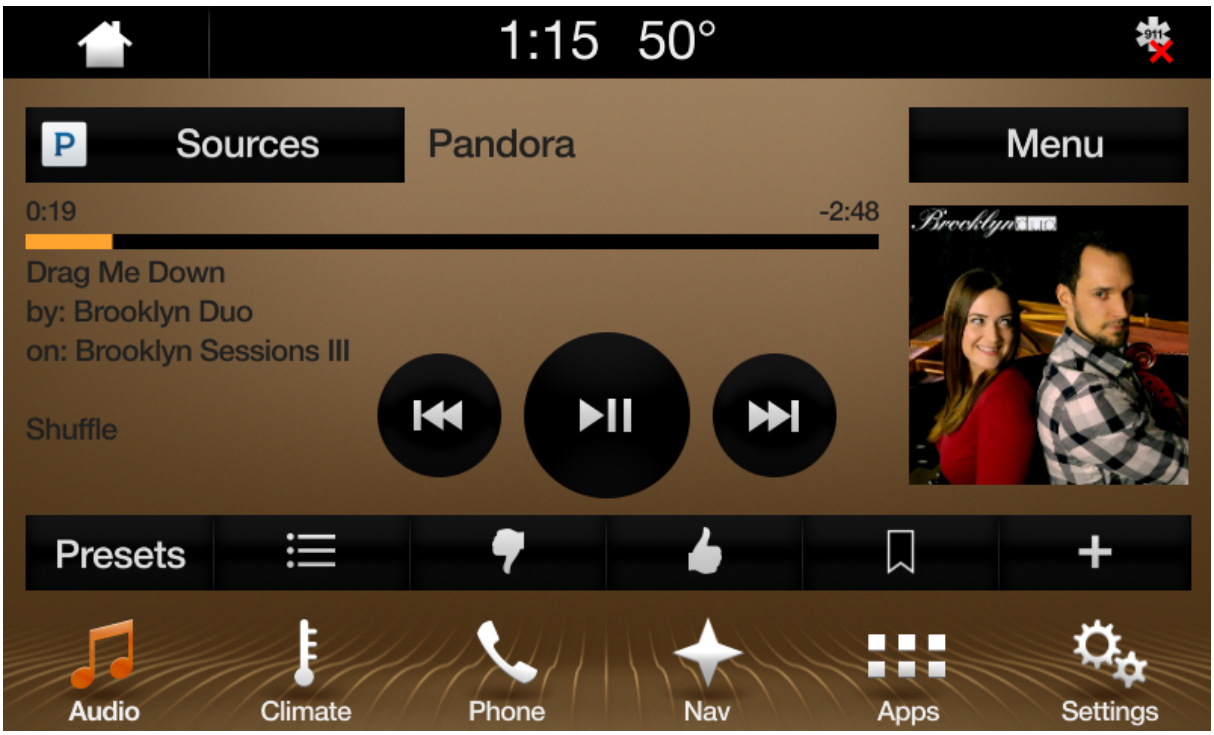
Your app should display things like song name, artist name, and optionally the album name. It is encouraged to use the following layout of information on the screen:

TEXTFIELD	INFORMATION
mainField1	Song
mainField2	Artist
mainField3	Album

It is encouraged to show static images of the album art, however moving or interactive images are not allowed neither is the display of lyrics.

Add Link to DisplayInformation

Below is an example of the Pandora home screen using the MainFields and displaying a graphic.



USING PRESETS

If your app features different stations or streams you should consider allowing the user to save these stations to preset buttons. To receive notifications about preset button presses please use the `subscribeButton` RPC with the respective presets as arguments. On some SDL implementations you will be able to change the text on a preset button. To change this text use the `customPresets` array in the `show` RPC.

MANAGING AUDIO

When you receive the `NOT_AUDIBLE` state you should pause the audio and resume when you receive `AUDIBLE`. The pausing and resuming should be independent on the current `HMILevel` state.

SmartDeviceLink FAQ

Here are a few of the most common questions new developers have around the SmartDeviceLink project.

Is WiFi a supported transport?

The WiFi transports currently supported are for testing and debugging only; they are not production ready. If this feature is desired, a proposal can be introduced in the SDL Evolution Process.

Can I implement custom HMI templates?

This is possible, but not recommended as any app that builds itself for the head unit's custom template would not work with any other systems. If new templates are desired, they should go through the SDL Evolution Process and be adopted by the project.

Can I implement custom vehicle data messages?

This is possible, but not recommended as any app that builds itself for the head unit's custom vehicle data types would not work with any other systems. If new

vehicle data types are desired, they should go through the SDL Evolution Process and be adopted by the project.

What is the process for obtaining an App ID?

To obtain an App ID, you must first register for an account on the SDL Developer Portal, and then register the company to which the app should belong. Once your company is registered, you can select the App IDs tab from your company profile and click the "Create New App ID" button to provide your app information and obtain your App ID.

Why does SDL require that I use templates for non-navigation applications?

Templates are the best way for you to design your application with the driver's safety in mind. For more information on the benefits of templates, please see this document.

I didn't find the answers I was looking for, where else can I look?

Each project has documentation, guides, and may have their own FAQ that can help. If you still need help, please join the SmartDeviceLink Slack team and ask your question.