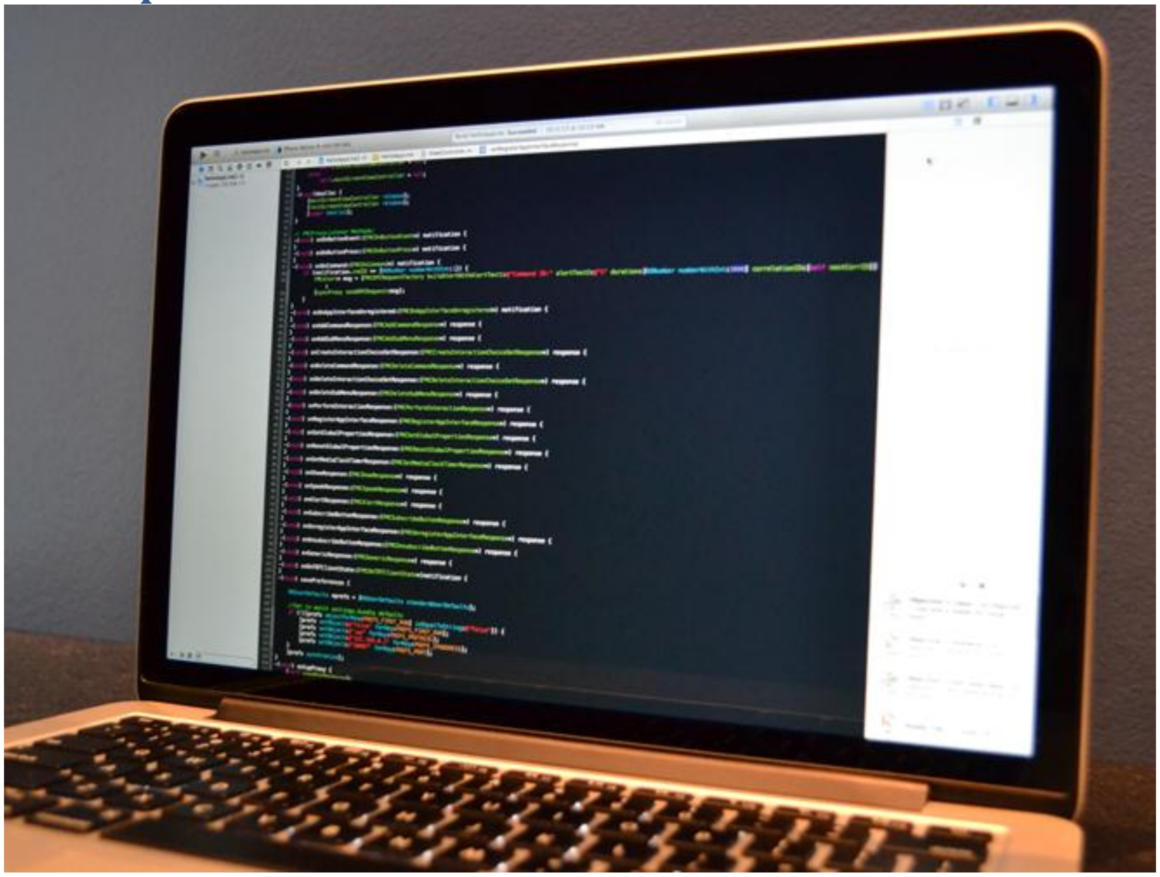
Contents

Develop



Now that you understand the concepts around interacting with the driver, ensuring safe experiences, and requirements that you must adhere to, you are ready to dive into the code and how it works.

In this section, you will learn about:

- What is the AppLink™ Library – Learn what it is and how to manage it.
- Lockscreen – Understand why it is important and what can and cannot be included.
- Remote Procedure Call (RPC) – Brief overview of the messages sent between your application and the system
- API Reference – Definition of anything and everything including each method, enumeration, and notification.
- Register Your Application – 'AppID': What it is, why you need it, and the form you need to fill out and submit in order to receive one.
- Downloads – Links to the mobile proxy libraries, release notes, and sample applications.
- Tutorials – Step-by-step guides that will take you from creating the connection to the system to advanced RPC combinations.

- <u>Upgrading Your Library</u> – Everything you need to know for those that already have an AppLink™-enabled application.
- <u>Tools</u> – Additional programs to aid in your development such as the AppLink™ Emulator.
- <u>Frequently Asked Questions (FAQ)</u> – Quick reference to common questions during your development process.

What is the AppLink™ Library?

The AppLink™ library is the intermediary that marshals information between your app and the vehicle. The AppLink Library directly interfaces with the HMI and uses Remote Procedure Calls (RPCs) and callbacks for communication (e.g. display text, speak a phrase, start audio capture, button-pushes, completion of asynchronous operations, etc.).

With the library version 1.6.0 on Android, we introduced new connection functionality on within the AppLink Library, and it is called Automated Lifecycle Management, or ALM. Notice that version 2.0 on iOS will not have ALM support.

Version Matrix

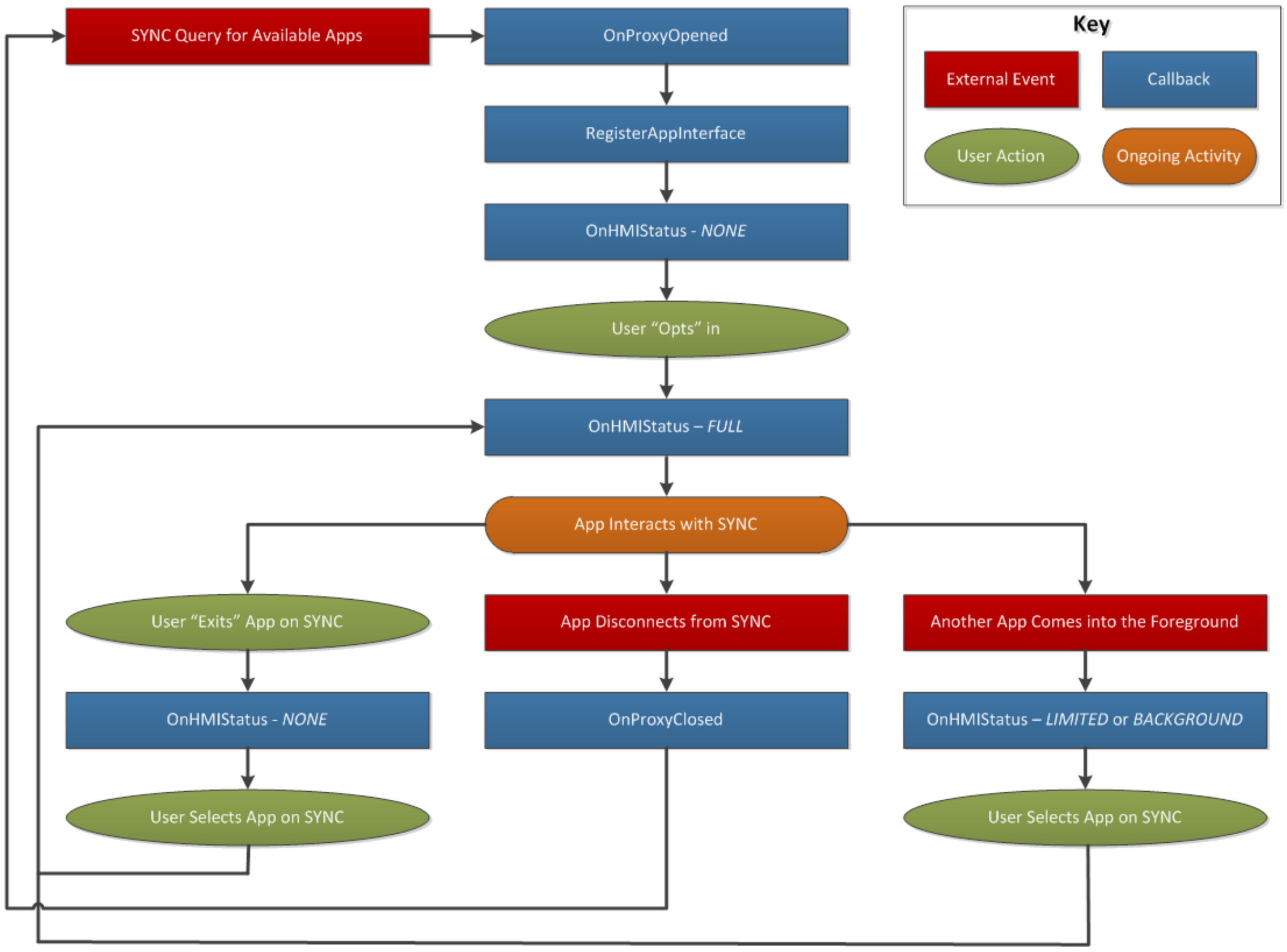
The below table depicts the availability of AppLink™ features in the various released proxies.

AppLink™ Version	Android Proxy	iOS Proxy
1.0	1.6.1	1.5.3
1.1	1.6.1	1.5.3
2.0	2.0	2.0

Legacy SyncProxy Lifecycle Management (iOS)

It is very important to manage the legacy AppLink Library correctly in order for an application to always show up in the Mobile Applications menu and be available for users to select. Failure to manage the lifecycle of your proxy as described by Ford Motor Company will result in undesired behavior. This will not only affect the functionality of your application, but will fail the validation tests required to have your application approved.

The entire lifecycle can be summarized by the figure below:



Legacy AppLink Library Lifecycle Management

Once a AppLink Library object has been created and used in vehicle by the user, it needs to be closed and discarded (i.e. when the user leaves their vehicle, quits the app on the device, etc.). Simply said, the AppLink Library is not reusable.

Connecting AppLink Library

Though the AppLink Library exposes a single initializer for instantiation, the simplest way to create an AppLink Library object is by using the *SyncProxyFactory*. The *SyncProxyFactory* provides methods to build a SyncProxy object using the default transport and protocol configuration, allowing the object to be constructed simply.

When building a SyncProxy instance, the application must provide an implementation of the *IProxyListener* interface. The app's proxy listener will be the object that receives callbacks when SYNC® sends responses and notifications to the app.

Building a AppLink Library instance enables the application to be found by SYNC®, but does not mean that the application is immediately connected. SYNC® drives the initial connection to the device in response to user interaction in the vehicle. The application will know SYNC® is connected when it receives the *onProxyOpened* callback on its proxy listener.

When the application is notified the SyncProxy is open, it should register by sending a *RegisterAppInterface* request as quickly as possible (ideally in the *onProxyOpened* callback method). If the application does not successfully register within 20 seconds, the connection will be closed. When the application receives an OnHMIStatus notification with *HMI Level* not equal to NONE, the application may send additional requests to SYNC®.

Disconnecting SyncProxy

There are two ways a registered application can disconnect from SYNC®:

- Send an *UnregisterAppInterface* request.
    - When the corresponding response is received, the application should then call the *dispose* method. If the application closes the SyncProxy before receiving the *UnregisterAppInterface* response, the user will be notified of an error that occurred, via a spoken notification.
- If the connection between SYNC® and the application is unexpectedly lost (e.g. the mobile device and vehicle are out of range of each other, etc.), the application will be notified via the *onProxyClosed* callback.

After SYNC® and the application are disconnected; the application should call the *dispose* method.

Reconnecting SyncProxy

Due to the lifecycle of the SyncProxy, reconnecting the application with SYNC® requires a new SyncProxy object to be instantiated. Once the SyncProxy is disconnected, it should be disregarded, and a new SyncProxy should be instantiated for reconnection.

Multiple SyncProxy Instances

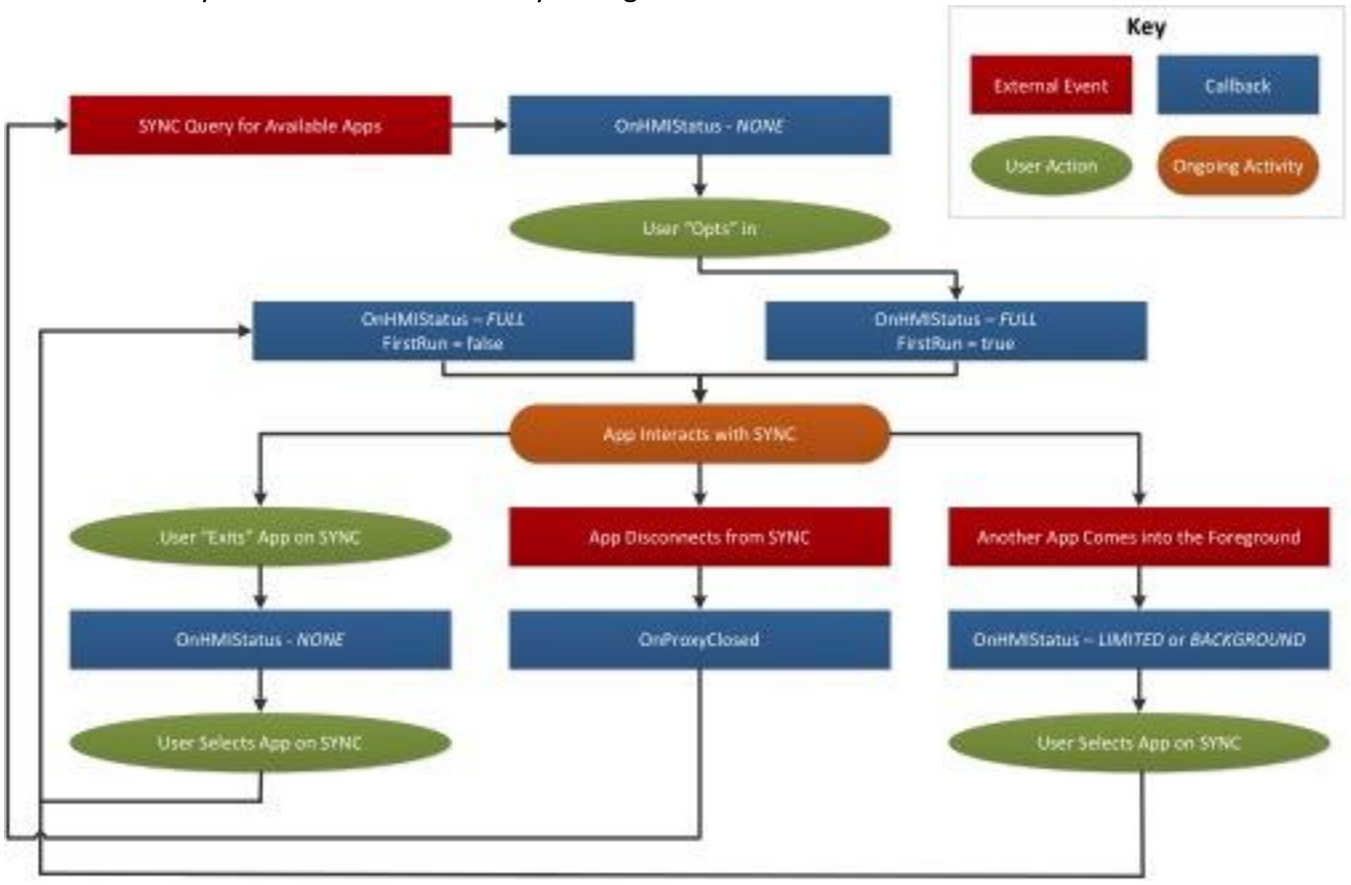
An application cannot instantiate more than one concurrent SyncProxy objects, though SyncProxy objects can be instantiated serially. This will make the application unusable by the user.

Automated Lifecycle Management (ALM) (Android)

When using the ALM version of the SyncProxy, SyncProxyALM, the SyncProxyALM manages the lifecycle of the connection to SYNC® on behalf of your application.
Once the SyncProxyALM object has been instantiated, your application simply needs to monitor and respond to lifecycle callbacks on the IProxyListenerALM interface, or the app's proxy listener.

It is very important to monitor and respond to the lifecycle events in the IProxyListenerALM, and failure to do so will result in undesired behavior. This will not only affect the functionality of your application, but you will fail the validation tests required to have your application approved.

The entire lifecycle can be summarized by the figure below:



SyncProxyALM Lifecycle

The SyncProxyALM object only needs to be disposed of and discarded when your application no longer requires further communication with SYNC® or an unrecoverable error occurs. Otherwise, the SyncProxyALM object will manage the connections to and subsequent disconnections from any AppLink™ enabled SYNC® units for the duration of its lifecycle.

Connecting the SyncProxyALM

The SyncProxyALM object exposes several constructors for instantiation. Through these constructors, your application can pass in important information, such as your application's name, which the SyncProxyALM will use to register an interface with SYNC® on behalf of your application.

When building a SyncProxyALM instance through the most basic constructor, your application must provide an implementation of the IProxyListenerALM, its name, an appId, and a Boolean to indicate whether or not your application is a media application, and an implementation of the IProxyListenerALM interface. This IProxyListenerALM interface will be the object that receives callbacks when SYNC® sends responses and notifications to your application.

Building a SyncProxyALM instance enables your application to be found by SYNC®, but does not mean that your application is immediately connected. SYNC® drives the initial connection to the device in response to user interaction in the vehicle. Your application will know SYNC® is connected when it receives an onHMIStatus callback on its proxy listener.

Disconnecting the SyncProxyALM

There are three ways a registered application can disconnect from SYNC®:

- Requesting a Disconnection
- Unexpected Loss
- Unrecoverable Error

Requesting a Disconnection

One way to request a disconnection is by calling the resetProxy method. When this method is called, the SyncProxyALM disconnects from any existing connection with SYNC®, disposes of all transient resources, and reinitialize itself. Once the re-initialization has completed, the proxy will be able to accept new connections from SYNC®.

If your application calls the other disconnection method, the dispose method, the SyncProxyALM permanently disconnects form any existing connection with SYNC®, disposes of all resources. At this point, the SyncProxyALM object can be discarded. A new SyncProxyALM object will need to be instantiated before further communication with SYNC® can occur.

Unexpected Loss

If the connection between SYNC® and your application is expectedly or unexpectedly lost (e.g. the mobile device and vehicle are out of range of each other, etc.), your application will be notified via the OnProxyClosed callback receiving SYNC_PROXY_CYCLED exception. The SyncProxyALM will continue to monitor for new SYNC® connections.

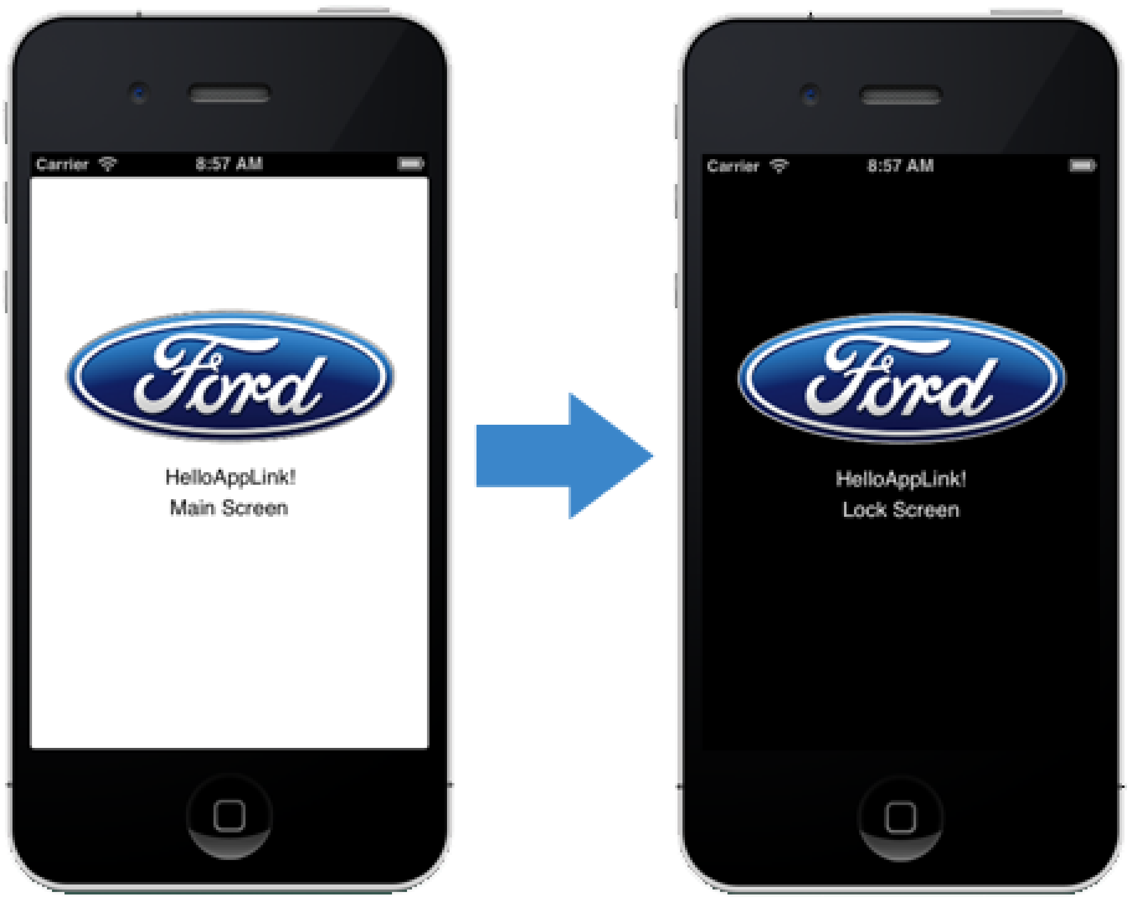
Unrecoverable Error

If an unrecoverable error, which cannot be resolved without intervention by your application, occurs in the proxy, your application will be notified via the OnProxyClosed callback. At this point, the SyncProxyALM object must be disposed of and discarded. A new SyncProxyALM object will need to be instantiated before further communication with SYNC® can occur.

Multiple SyncProxyALM Instances

An application cannot instantiate more than one concurrent SyncProxyALM objects, though SyncProxyALM objects can be instantiated serially. This will make your application unusable by the user.

Lockscreen



General Requirement

As per Ford Motor Company's driver distraction rules, we require any AppLink™ application to implement a lockscreen that must be enforced while your application is active on the system while the vehicle is in motion.

This lockscreen must perform the following:

- Limit all application control usability from the mobile device with a full-screen static image overlay or separate view.

For simplicity, we recommend implementing the lock screen when you receive a HMI Level of FULL, LIMITED, or BACKGROUND.

Driver Distraction Notification (onDriverDistraction)

Alternatively, you can listen for the onDriverDistraction notification, which signals when the vehicle is moving over a certain speed limit. If your application listens for the onDriverDistraction notification, it may unlock and lock the screen appropriately based on the value returned in the notification. Most importantly, the act of locking of the screen must be instantaneous and not delayed due to any other application processing or threading.

If you choose to listen for the onDriverDistraction notification, your application must enable the lockscreen when the following conditions are met:

- HMIStatus of FULL and DriverDistractionState of DD_ON
- HMIStatus of LIMITED and DriverDistractionState of DD_ON
- HMIStatus of BACKGROUND and DriverDistractionState of DD_ON and application has been selected / launched
- Very first HMIStatus of FULL
    - **Note:** Your application must show the lockscreen upon receiving the first HMIStatus of FULL regardless of the DD_MODE. Please see “Older Vehicle Software Versions” for more info.

Older Vehicle Software Versions

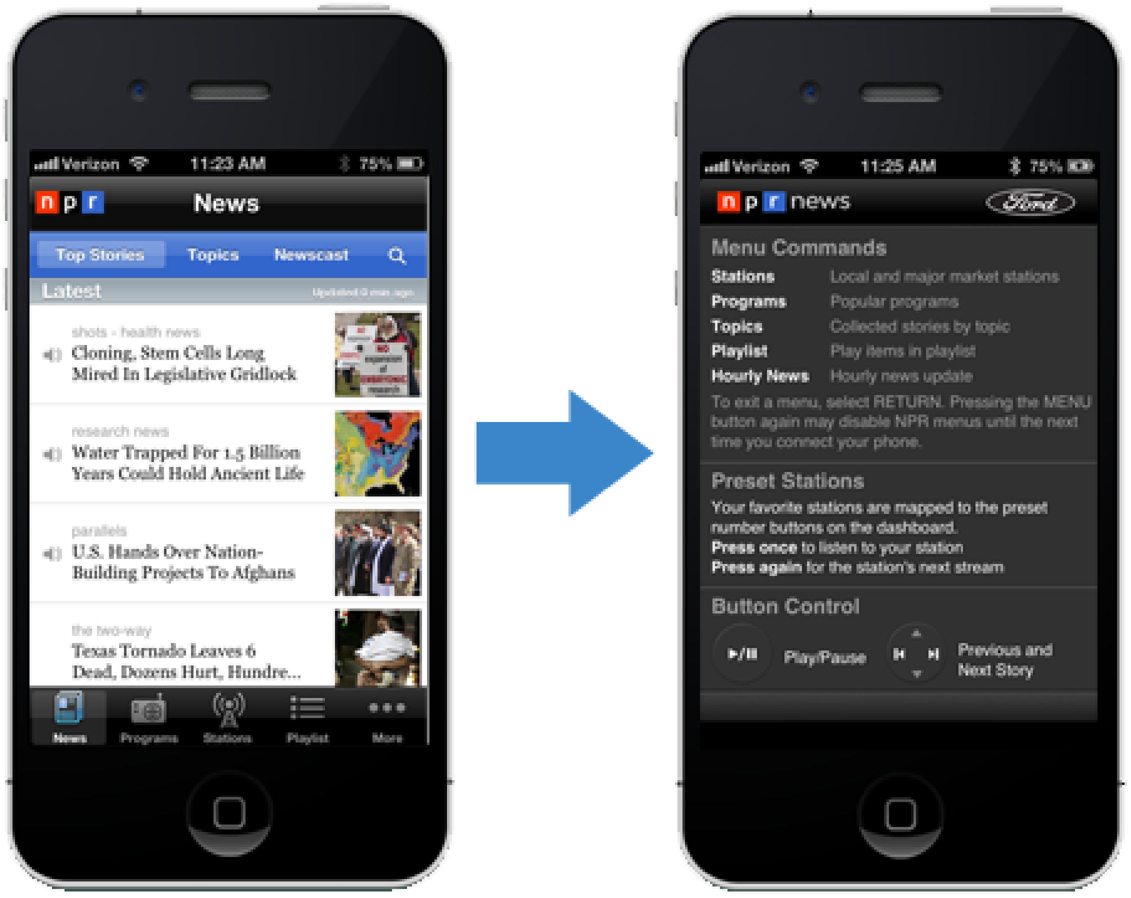
On older vehicle software versions (approx. 2012 and older), your application will not receive the state of the driver distraction mode upon registering, though this happens on newer software versions. Therefore, your application must show the lockscreen upon receiving the first HMIStatus of FULL regardless of the DriverDistractionState. This implementation protects for the scenario of starting the app (first FULL notification) while the vehicle is being driven.

Android™ Requirements

For Android™, the following phone menu options and physical buttons must also be limited:

- Quit or Exit Application Menu Item
    - If implemented, this should stop any audio playback, unregister the application from the SYNC®, and execute your normal quitting procedure.
- Back button
    - While the lockscreen is up, you must detect this event and not allow the lockscreen to disappear from the UI. It is acceptable if the phone’s UI returns to the home screen or any screen outside of your application’s context.

Additional Functionality



shots - health news
Cloning, Stem Cells Long Mired In Legislative Gridlock

research news
Water Trapped For 1.5 Billion Years Could Hold Ancient Life

parallels
U.S. Hands Over Nation-Building Projects To Afghans

the two-way
Texas Tornado Leaves 6 Dead, Dozens Hurt, Hundre...

Menu Commands

Stations	Local and major market stations
Programs	Popular programs
Topics	Collected stories by topic
Playlist	Play items in playlist
Hourly News	Hourly news update

To exit a menu, select RETURN. Pressing the MENU button again may disable NPR menus until the next time you connect your phone.

Preset Stations

Your favorite stations are mapped to the preset number buttons on the dashboard.
Press once to listen to your station
Press again for the station's next stream

Button Control

Your application can add functionality such as, but not limited to, the following:

- Disconnect button
  - It is not recommended to include a disconnect button on an app's lockscreen and/or use the UnregisterAppinterface RPC. Apps will automatically disconnect when the device is unplugged.
  - If implemented, this should unregister the application from SYNC®, close and dispose the SyncProxy, re-create it, and then finally clear the lockscreen.
- Informative lockscreen
  - This can be a quick reference for high-level voice commands, button controls, etc.
- Location (or other relevant information) on a map
  - if implemented, a user should not be able to modify or browse the map on the device itself

Any and all additional lockscreen functionality will need to abide by all driver distraction rules and be approved by Ford Motor Company before the application is approved for app store submission.

Remote Procedure Call (RPC) Overview

An AppLink™ RPC is a request/response pair that instructs AppLink™ to perform some HMI operation, such as displaying text, speaking text, or to define other elements used in HMI communication, such as commands.

Requests

Requests are made by the application and require that the application provide various parameters, including a correlationID.

Responses

All responses include at least four data elements:

- CorrelationID
- Success
- Result Code
- Info

CorrelationID

CorrelationID is an integer value that connects a specific request to the specific response. The correlationID is returned to the application as a parameter in the corresponding response. It is the responsibility of the application to ensure that the correlation IDs are unique, so that a given response can be unambiguously associated with its precipitating request, if the application requires it.

A given correlationID value cannot be used in two or more concurrently active requests. That is, if a given correlationID value is provided in one operation request (e.g. Speak), then the same correlationID value cannot be used in another operation request until the first operation request completes.

Success

Success is a Boolean indicating whether the original request was successfully processed.

ResultCode

ResultCode is an enumeration that provides additional information about a response returning a successful or failed outcome.

Info

Info is a string of text representing additional information returned from SYNC. This is useful in debugging various RPC requests.

GenericResponse

If SYNC® does not recognize an RPC request (i.e. it is not one of the RPC operations described in this section), SYNC® will send a GenericResponse RPC response. The GenericResponse RPC response contains the same parameters that are in any other RPC response; correlationID, success, resultCode, and info.

HMI Level Requirements for RPC

SYNC® will take in requests from an application at any given time, but can only process them based on the current Connection State and HMI Level of your application.

For instance, if your application wants to send a PerformInteraction request, your application needs to have an HMI Level of FULL in order for SYNC® to successfully process and execute the request. The chart below can be used as a quick reference guide for the HMI Level requirements of each RPC request.

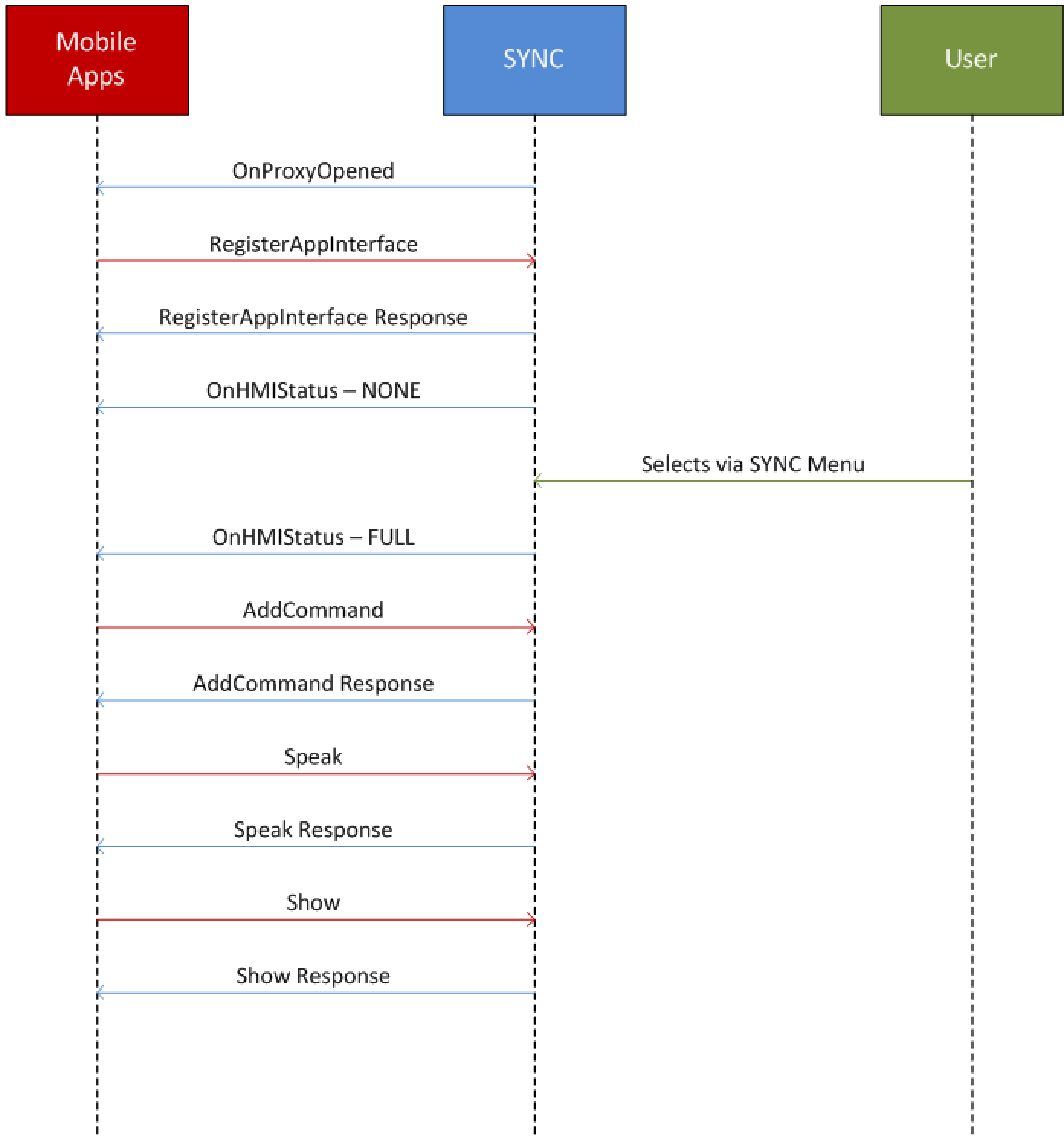
RPC Request Availability Matrix						
SYNC Connection State	**State**	**Connected**				**Not Connected**
	Callback	OnHMIStatus				OnProxyClosed
HMI Level		FULL	LIMITED	BACKGROUND	NONE	N/A
RPC Request		AddCommand			▨	▨
		AddSubMenu			▨	▨
		Alert		if fct group Notification	▨	▨
		AlertManeuver			▨	▨
		ChangeRegistration			▨	▨
		CreateInteractionChoiceSet			▨	▨
		DeleteCommand			▨	▨
		DeleteInteractionChoiceSet			▨	▨
		DeleteSubMenu			▨	▨
		EndAudioPassThru		▨	▨	▨
		EncodedSyncPData			▨	▨
		GetDTCs			▨	▨
		GetVehicleData			▨	▨
		PerformAudioPassThru		▨	▨	▨
		PerformInteraction	▨	▨	▨	▨
		ReadDID			▨	▨
		ResetGlobalProperties			▨	▨
		ScrollableMessage			▨	▨
		ShowConstantTBT			▨	▨
		SetGlobalProperties			▨	▨
		SetMediaClockTimer			▨	▨
		Show*			▨	▨
		Slider			▨	▨
		Speak		▨	▨	▨
		SubscribeButton			▨	▨
		SubscribeVehicleData			▨	▨
		UnregisterAppInterface				▨
		UnsubscribeButton			▨	▨
		UnsubscribeVehicleData			▨	▨
		UpdateTurnList			▨	▨

Notes
* show() can be performed whenever the Sync Interface is available, however this results won't be visible to the user until the application has an HMI level of either *FULL* or *LIMITED*

RPC Request Availability Matrix

Typical RPC Exchange

The following diagram demonstrates a typical sequence of RPC exchanges for a simple application:



Typical RPC Exchange

Here, an application registers the application interface with SYNC®, populates a menu with a command, and speaks a welcome message.

Note: the diagram assumes that a transport connection and a transmission protocol RPC session have been established, and that RPCs are being marshaled appropriately.

Functional Grouping of RPCs

The RPCs are grouped by function, and when applying for an appID, the desired RPC functional groups need to be selected.

RPC functional groups are:

- Base
  - AddCommand
  - AddSubMenu
  - Alert (state HMI_FULL or HMI_LIMITED)
  - ChangeRegistration
  - CreateInteractionChoiceSet
  - DeleteCommand
  - DeleteInteractionChoiceSet
  - DeleteSubMenu
  - EncodedSyncPData
  - GenericResponse
  - PerformInteraction
  - RegisterAppInterface
  - ResetGlobalProperties
  - SetGlobalProperties
  - SetMediaClockTimer
  - Show
  - Speak
  - SubscribeButton
  - UnregisterAppInterface
  - UnsubscribeButton
  - ScrollableMessage
  - PerformAudioPassThru
  - EndAudioPassThru
- Navigation
  - ShowConstantTBT
  - AlertManeuver
  - UpdateTurnList
- Vehicle Information
  - RPCs:
    - GetVehicleData
    - SubscribeVehicleData
    - UnsubscribeVehicleData
  - parameters, available to all above RPCs:
    - externalTemperature
    - fuelLevel

        - fuelLevelState
        - instantFuelConsumption
        - odometer
        - tirePressure
        - vin
- Driving Characteristics
    - RPCs:
        - GetVehicleData
        - SubscribeVehicleData
        - UnsubscribeVehicleData
    - Parameters, available to all above RPCs::
        - beltStatus
        - driverBraking
        - prndl
        - rpm
- Location
    - RPCs:
        - GetVehicleData
        - SubscribeVehicleData
        - UnsubscribeVehicleData
    - Parameters, available to all above RPCs::
        - gps
        - speed
- Notification
    - Alert (being in the state HMI_BACKGROUND)

Requesting app permissions from Ford

Ford has introduced a system that enables remote app management, including the ability to disable apps running on SYNC and the ability to restrict app functionality.

For consent and usage purposes, RPCs are bundled into functional groups. When requesting an appID, a developer has the opportunity to request access to certain functional groups. Please see "Remote Procedure Call (RPC)" for a list of functional groups and their RPCs.

In addition to Ford approval, certain app permissions also require user consent in vehicle. The first time your app is selected in vehicle, the user will be prompted for any required consent per legal guidelines. The user's initial response is all or nothing. However, individual app permissions can be enabled or disabled at any time, while not driving, in the vehicle Mobile Apps Settings Menu.

AppID

Ford issues appIDs to developers via email or website account after reviewing the "AppID Request Form." The app must pass this appID when creating a SyncProxy. See "Changing app permissions" for additional information. For example, if the appID = 438316430:

```
proxy = new SyncProxyALM(this,"Hello AppLink",true,"438316430");
```

During the development phase, your appID will only work with TDKs and a limited number of vehicles as specified in your request. Once the app is validated by Ford, Ford will roll the appID into production, making it compatible with all AppLink-enabled Ford vehicles.

Downloads

a. Content from each current iOS / Android SDK page
    i. Proxy (Heading 2)
        1. Links for each proxy separated by mobile OS
        2. Text for the release notes under each as they may differ
    ii. Sample Applications (Heading 2)
        1. Links for each sample application separated by mobile OS
        2. Text under each that explain the application and what it does

b. Content Owner: Ed W. and EU/APA counterparts

Tutorials

Ford cannot stress the following point enough; if you are new with AppLink™, the vehicle HMI, or even new with developing, following our tutorials will get you up to speed and walk you through the most common pitfalls.

General

Launching an AppLink™ Application – Answers the most common question we receive, “How do I launch an app?”

Android™

Getting Started on Android™ - Takes you from configuring your IDE, setting up your Application Manifest, and beginning the integration of the AppLink™ libraries.

iOS

Getting Started on iOS - Takes you from configuring your Xcode, setting up your Build Settings, and beginning the integration of the AppLink™ libraries.

Upgrading Your Library

In order for your application to work with any new vehicle, you must update to AppLink SDK version 2.0. This guide assumes you already use the latest library files and lifecycle management techniques.

Android

Replace your Existing Library

Download the new library file (v2.1.1) to your file system and replace your existing SyncProxyAndroid.jar file (v1.6.1) in your project.

Once replaced, open your project in Eclipse and select **Project** > **Clean**. On the dialog that shows, ensure your project(s) is selected and click **OK**.

You also will need to replace the reference in the Build Path from the old library to the new.

Modifying your Proxy Object Creation

The proxy object has been updated to include some additional parameters, and the most important one is the additional of 'AppID'.

Implementing the Additional Methods

Once you have modified the creation of the Proxy Object, the final step is to implement the additional methods that AppLink™ 2.0 brings in.

With Android, this is extremely easy. Simply right-click on the Service class and click on the **Add unimplemented methods** link.

Additional Steps

One additional step you must perform is to modify the Manifest file to include the following lines:

```
<uses-permission android:name="android.permission.INTERNET" />
<!-- Required to check if WiFi is enabled -->
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

iOS

Replace your Existing Library

Download and unzip the new AppLink framework to your project folder.
Remove your existing library (.a) file from the project, and add the AppLink framework to the Linked Frameworks and Libraries section of the target by clicking the Add (+) button, then the “Add Other…” button and then select the “Applink.framework” folder.

Next, replace and condense all existing imports in your project files from #import “FM<Class>.h” to a single #import <AppLink/AppLink.h> as this will import all the required headers needed for AppLink.

Finally, replace all uses of “FM<Class>” class names with “FMC<Class>”, i.e.: FMShow to FMCShow (A case sensitive Find and Replace works very well here).

Modifying your FMCRegisterAppInterface Request

The FMCRegisterAppInterface request has been updated to include some additional required parameters, the most important of which are “appID” and “languageDesired”.

These have also been added to the FMCRPCRequestFactory’s buildRegisterAppInterfaceWithAppName method and will need to be added if the FMCRPCRequestFactory is used.

Removing Unused FMCProxyListener Callbacks

The new AppLink framework has made the majority of the FMCProxyListener callbacks optional. There is no need to implement them if your app does not use them. If they were in your app as required placeholders before, they can now be removed.