



# **Wrong Way Driver Warning SDK Documentation iOS 3.1.2**

Robert Bosch GmbH

November 7, 2018

## Contents

<b>I. Changelog</b>	<b>3</b>
<b>II. Integration How To</b>	<b>7</b>
<b>III. Error Code</b>	<b>15</b>
<b>IV. HMI Guidelines</b>	<b>17</b>
<b>V. FAQ</b>	<b>26</b>
<b>VI. Best Practices</b>	<b>30</b>
<b>VII. Quick Verification Test</b>	<b>32</b>
<b>VIII. Disclosure Document</b>	<b>35</b>



# Part I. Changelog



## 1. WDW-SDK Version 3.1.2

### 1.1. Important changes

- ▶ Bitcode support has been activated for the WDW-SDK.

### 1.2. Bug fixes

- ▶ The robustness of the WDW-SDK has been improved.

## 2. WDW-SDK Version 3.1.0

### 2.1. Important changes

- ▶ The WDW-SDK has been extended to support additional countries.

## 3. WDW-SDK Version 3.0.0

### 3.1. Important changes

- ▶ The amount of errors the WDW-SDK generates has been reduced. All errors have been renamed.

### 3.2. Bug fixes

- ▶ The robustness of the WDW-SDK has been improved.

### 3.3. New behavior

- ▶ The error handling of the WDW-SDK has been adjusted. Every reported error will automatically stop the monitoring. It is the responsibility of the Vendor App to restart the monitoring.
- ▶ The WDW-SDK will enter its power saving state a lot earlier.
- ▶ The areas in which the WDW-SDK will collect data have been refined.

## 4. WDW-SDK Version 2.2.0

### 4.1. Bug fixes

- ▶ The robustness of the WDW-SDK has been improved.
- ▶ The GPS update algorithm has been improved.



## 4.2. New behavior

- ▶ The WDW-SDK sets an upper limit to the amount of data provided by the `didReceiveNewData` callback.

## 5. WDW-SDK Version 2.1.1

### 5.1. Bug fixes

- ▶ A bug introduced with the optimized power consumption has been fixed.

## 6. WDW-SDK Version 2.1.0

### 6.1. Important changes

- ▶ Third-party code dependencies have been removed.
- ▶ Start-up diagnostic has been introduced. It verifies the integration of the WDW-SDK and provides log information for troubleshooting.
- ▶ The power consumption of the WDW-SDK has been reduced.
- ▶ The WDW-SDK start-up performance has been improved.
- ▶ New error codes have been introduced:
  - ▶ `WDW_UnsupportedHardware_Error`: raised when attempting to start monitoring on a device with no GPS sensor installed.
  - ▶ `WDW_PushDeregistration_Error`: raised when the device fails to deregister from the push server.
- ▶ The following error codes have been deprecated; handling is no longer needed:
  - ▶ `WDW_Json_Generation_Error`
  - ▶ `WDW_BB_Conf_Error`
  - ▶ `WDW_BB_General_Error`
  - ▶ `WDW_BB_Empty_Map`
  - ▶ `WDW_BB_Wrong_Location`

### 6.2. Bug fixes

- ▶ The WDW-SDK will no longer crash on multiple incoming push messages.
- ▶ The log warning 'UI API called on background thread' has been fixed.



## 7. WDW-SDK Version 2.0.0

### 7.1. Important changes

- ▶ The following compile time dependencies have been removed:
  - ▶ Apple Reachability OSS
  - ▶ AeroGear Push LibraryCredits and license information on these dependencies are no longer needed.
- ▶ The main entry point to the WDW-SDK has been changed from `WrongwayDriverWarningSDKMain` to `WDWClient`.
- ▶ The main callback interface has been changed from `WrongwayDriverWarningSDKDelegate` to `WDWClientDelegate`.
- ▶ The method to initialize the WDW-SDK has been improved: the class `WDWBuilder` is used to create an instance of the `WDWClient`.
- ▶ The `WDWSDKAvailability` options have been renamed. The following states are used for availability:
  - ▶ `WDWClientUnavailable`
  - ▶ `WDWClientTemporarilyUnavailable`
  - ▶ `WDWClientAvailable`
- ▶ The following Error items have been removed:
  - ▶ `WDW_Json_ObservedData_Error`
  - ▶ `Json_ObservedData_Error`
  - ▶ `WDW_Logic_UUID_Error`
  - ▶ `WDW_Service_Not_Started`
  - ▶ `Push_Disabled_Error`
  - ▶ `Missing_Internet_Permissions`

### 7.2. New behavior

- ▶ Logging was introduced to the WDW-SDK. The log can be controlled by four different log levels defined by `WDWLogLevel` type.
- ▶ The log output of the WDW-SDK can be defined using the `setLogLevel` method of the class `WDWBuilder`.
- ▶ The integrated push functionality can be deactivated by omitting the `withPushConfiguration` method of the class `WDWBuilder`.
- ▶ The integrated message evaluation can be replaced by a custom implementation of the `WDWMessageEvaluator` interface to the `withMessageEvaluator` method of the class `WDWBuilder`.



## Part II. Integration How To



## 1. Introduction

The WDW-SDK is the client part of the WDW service. Once it is integrated into your application, it provides the functionality to collect device sensor data and receive wrong way driver warnings. The following document explains how to integrate the WDW-SDK. For further information about specific functionality, refer to the provided [HTMLDocs](#).

## 2. Preconditions

In order to successfully integrate the WDW-SDK, the following preconditions have to be met:

- ▶ Apple computer running Mac OS X 10.12 or higher with Xcode 8.0 or higher installed
- ▶ iOS application targeted for iOS 8 or higher
- ▶ iOS application code written in Objective-C or Swift
- ▶ Identifiers and push credentials provided to you by Bosch (See [FAQ - iOS](#))

## 3. Integration

The WDW-SDK is built in a modular manner. The following section explains the default integration. Section [3.3](#) explains further integration options.

### 3.1. Project Setup

To set up your project to compile with the WDW-SDK, the following steps are needed:

1. In the *General* tab of your application's target, add the `WrongwayDriverWarningSDK.framework`:

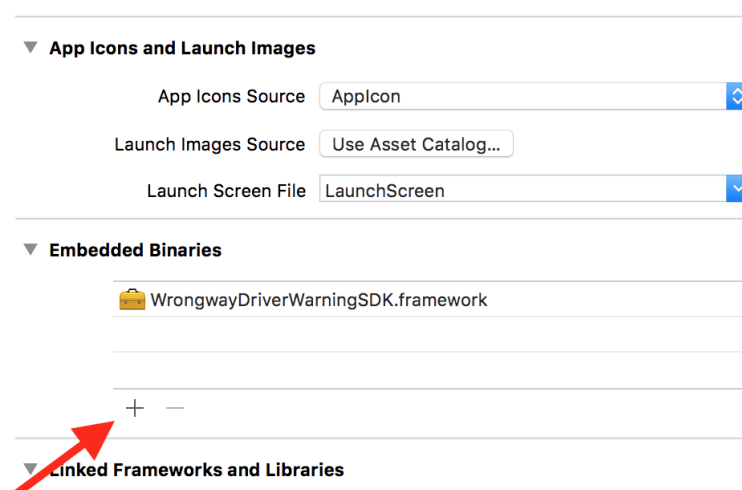


Figure 1: Embedding the WDW-SDK



2. Location services need to be switched on for the WDW service to be able to detect wrong way drivers. For this reason, in the *Info* tab of your application’s target, add values to the following keys explaining to your users why your application needs location tracking:

- ▶ NSLocationWhenInUseDescription
- ▶ NSLocationAlwaysUsageDescription

▼ Custom iOS Target Properties

Key	Type	Value
NSLocationWhenInUseUsageDe...	String	GPS access required to
CFBundleIdentifier	String	\$(PRODUCT_BUNDLE_I
CFBundleInfoDictionaryVersion	String	6.0
UIMainStoryboardFile	String	Main
CFBundleVersion	String	1
NSLocationAlwaysUsageDescription	String	GPS access required to
UIBackgroundModes	Array	(2 items)
CFBundleExecutable	String	\$(EXECUTABLE_NAME)

Figure 2: Custom iOS target properties

3. In the *Capabilities* tab of your application’s target, enable Push Notifications:

PROJECTWDWS...TARGETSWDWS...

▶ iCloudOFF

▼ Push NotificationsON

Steps: ✓ Add the Push Notifications feature to your App ID.  
✓ Add the Push Notifications entitlement to your entitlements file

▶ Game CenterOFF

▶ WalletOFF

▶ SiriOFF

▶ Apple PayOFF

▶ In-App PurchaseOFF

Figure 3: Enabling Push Notifications



4. In the *General* tab of your application's target, find the *Background Modes* section and make sure that the following options are checked:

- ▶ Location updates
- ▶ Remote notifications

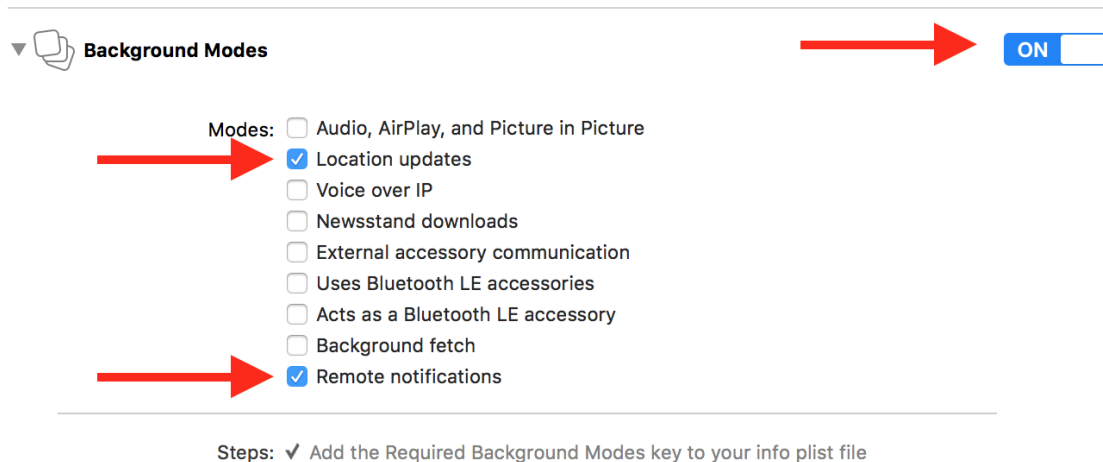


Figure 4: Background execution modes

### 3.2. SDK Default Instantiation

Once the WDW-SDK has been added as dependency, an instance needs to be created in order to use and control the functionality of the WDW-SDK. The following section explains how to create an instance and how to control and communicate with the WDW-SDK.

1. Import the WrongwayDriverWarningSDK headers:

```
#import <WrongwayDriverWarningSDK/WrongwayDriverWarningSDK.h>
```

Listing 1: Import headers

2. Define a property to hold the instance of WDWClient:

```
@property (nonatomic, strong) WDWClient *wdwClient;
```

Listing 2: Define a property for WDWClient instance

3. Implement the WDWClientDelegate protocol:

```
@interface MyAppDelegate() <WDWClientDelegate>
...
- (void)wdwClient:(WDWClient *)client didEnterMonitoredArea:(NSString *)areaId {
    // optional callback: get ready to receive data from the SDK
    // e.g. establish server connection
}

- (void)wdwClient:(WDWClient *)client didLeftMonitoredArea:(NSString *)areaId {
    // optional callback: get ready to stop receiving data from the SDK
    // e.g. close server connection
}
```



```

- (void)wdwClient:(WDWClient *)client didReceiveDrivingWarning:(WDWWarning *)warning {
    // display warning to user
}

- (void)wdwClient:(WDWClient *)client
    didChangeAvailability:(WDWClientAvailability)availability {
    // notify user of WDW-SDK availability
}

- (void)wdwClient:(WDWClient *)client didReceiveError:(NSError *)error {
    // handle errors
}

- (void)wdwClient:(WDWClient *)client didReceiveNewData:(NSString *)json {
    // send data to server
}

```

Listing 3: WDW-SDK callback methods

4. The WDW service utilises the *Apple Push Service* for delivering the wrong way driver events. Hence, all push notifications need to be forwarded to the SDK instance for evaluation:

```

- (void)application:(UIApplication *)application
    didReceiveRemoteNotification:(NSDictionary *)userInfo
    fetchCompletionHandler:(void (^)(UIBackgroundFetchResult))handler {
    if ([self.wdwClient evaluatePushMessage:userInfo] == nil) {
        NSLog(@"Incoming push message is not related to WDW");
        // handle application-related push message
    }
    handler(UIBackgroundFetchResultNewData);
}

```

Listing 4: Forwarding a push message to the WDW-SDK

5. A device token is required to allow the SDK to register itself with the WDW's own push service:

```

//Device token is required to start the SDK in it's default configuration
@property (nonatomic, strong) NSData *token;
...
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Make sure your application registers with the Apple Push Notification service
    [application registerForRemoteNotifications];
    return YES;
}

- (void)application:(UIApplication *)application
    didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken {
    // Save received token
    self.token = deviceToken;
    // Now ready to run the SDK
    [self buildSDK];
}

```

Listing 5: Gathering the device token



6. A `WDWPushConfig` object needs to be created with the credentials given by Bosch (See [FAQ - iOS](#)) and the device token obtained in the previous step:

```
- (WDWPushConfiguration *)pushConfiguration {  
    return [[WDWPushConfiguration alloc]  
        initWithPushToken:self.token  
        server:@"Put Push Server URL here"  
        variantID:@"Put Server Variant here"  
        secret:@"Put Server Secret here"  
    ];  
}
```

Listing 6: Creating `WDWPushConfiguration` object

7. The main entry to the WDW-SDK is the `WDWClient` interface. This interface is used to control the SDK; e.g. to start and stop it. A builder pattern is used to create an instance of the `WDWClient`:

```
- (void)buildSDK {  
    self.wdwClient = [[[[WDWBuilder new]  
        withFleetID:@"Put Fleet ID here"]  
        withPushConfiguration:[self pushConfiguration]]  
        build];  
  
    self.wdwClient.delegate = self;  
}
```

Listing 7: Building the `WDWClient` instance with push configuration

8. The WDW-SDK has two different states: started or stopped. When the WDW-SDK is started, it collects device sensor data and receives wrong way driver warnings. When it is stopped, it does nothing. To start the monitoring, use the following method call:

```
[self.wdwClient startMonitoring];
```

Listing 8: Start monitoring

9. The WDW-SDK collects data until it is specifically told to stop. Even if the app is in the background, the WDW-SDK continues to collect device sensor data and receive wrong way driver warnings. To stop the monitoring, use the following method call:

```
[self.wdwClient stopMonitoring];
```

Listing 9: Stop monitoring



### 3.3. SDK Advanced Instantiation Options

Since the WDW-SDK is modular, the WDW-SDK can be configured in different ways using the builder pattern described above. The following section describes the different options.

#### 3.3.1. Disabling Internal Push Module

If your application's infrastructure provides its own WDW notification delivery implementation, you can run the WDW-SDK without Push Module enabled:

```
- (void)buildSDK {
    self.wdwClient = [[[WDWBuilder new]
                      withFleetID:@"Put Fleet ID here"]
                     build];

    self.wdwClient.delegate = self;
}
```

Listing 10: Disabling internal push

Note: even if you use your own push notification delivery channel, you still need to forward incoming push messages to the SDK.

#### 3.3.2. Custom Message Evaluation

By default, the integrated message evaluation of the WDW-SDK is enabled. The WDW-SDK evaluates every forwarded push message whether it is relevant for the current driver or not. For this evaluation, different metrics are used, e.g. distance, heading or time. If this evaluation is not sufficient, it can be replaced by a custom message evaluation using the builder pattern. Use the following steps to implement a custom message evaluation:

1. Create a class that implements the `WDWMessageEvaluation` protocol. Implement the `evaluateMessage` method in order to react to defined `WDWWarning` messages based on the incoming data object:

```
@interface MyEvaluator : NSObject <WDWMessageEvaluation>
...
- (WDWWarning *)evaluateMessage:(NSDictionary *)object {
    ...
    warning.eventType = WDWEventThereIsWDW;
    return warning;
    ...
}
```

Listing 11: Creating custom evaluation



## 2. Register the MyEvaluator instance and pass it as WDWBuilder argument:

```
- (void)buildSDK {  
    MyEvaluator *evaluator = [[MyEvaluator alloc] init]  
  
    self.wdwClient = [[[[WDWBuilder new]  
        withFleetID:@"Put Fleet ID here"]  
        withMessageEvaluator:evaluator]  
        build];  
  
    self.wdwClient.delegate = self;  
}
```

Listing 12: Replacing the default message evaluator

For every incoming push message, the custom implementation will now be called to evaluate the message.

### 3.3.3. Diagnostics and Debug

The WDW-SDK provides logging at different log levels. All logs are output to a `stderr` stream. Four different log configurations are possible:

- ▶ `WDWLogLevelNone` - No log output.
- ▶ `WDWLogLevelError` - Only errors are logged.
- ▶ `WDWLogLevelWarning` - Errors and warnings are logged.
- ▶ `WDWLogLevelInfo` - Errors, warnings and additional information are logged.

By default, the WDW-SDK is set to `WDWLogLevelWarning`.

Changing the log level to `WDWLogLevelInfo`, for example, is done in the following way:

```
self.wdwClient = [[[[WDWBuilder new]  
    withFleetID:@"Put Fleet ID here"]  
    withLogLevel:WDWLogLevelInfo]  
    build];
```

Listing 13: Changing the log level



## Part III. Error Code



## 1. Introduction

The `WDWClientDelegate` returns few different errors when the behavior of the WDW-SDK is not as expected. All errors are delivered to the application over the `wdwClient:didReceiveError:` callback. Whenever an error occurs, the WDW-SDK will stop itself. The table below shows all possible errors given to the application. All of these errors need to be handled by the application.

## 2. List of errors

Name	Description
<b>WDW_Permissions_Error</b>	Error is thrown when the user does not grant or withdraws the permissions required by the WDW-SDK: Background execution or push notification permissions (only if WDW-SDK push is used).
<b>WDW_UnsupportedHardware_Error</b>	Error is thrown on starting the WDW-SDK when the device does not include GPS hardware.
<b>WDW_GPS_Access_Error</b>	Error is thrown on starting the WDW-SDK or in runtime when the user has disabled the Location Service or withdrawn the permission to use it.





## Part IV. HMI Guidelines



## 1. Introduction

The purpose of this document is to provide HMI guidelines for applications that integrate the Bosch WDW-SDK. The guidelines are applicable only to the HMI parts of the application relating to the WDW integration: the wrong way driver warnings sent to the application by the WDW service.

The document articulates the basics for designing in an automotive context and lists recommendations for implementing the WDW warnings in a driving relevant fashion.

All graphics shown here are for demonstration purposes only, to explain the concepts mentioned.

## 2. Guidelines

### 2.1. GUI, Layout & Text Requirements

There are two types of warnings that are shown to the driver. One, when the driver is driving his vehicle in the wrong direction on the motorway. Two, when the driver is driving in the correct direction on the motorway and is heading towards a wrong way driver:

- 1. I am a wrong way driver
- 2. I am in the vicinity of a wrong way driver

Necessary actions to be taken are prompted to the driver. The correct flow to be implemented inside the app is shown in the detailed flowchart below.

#### 2.1.1. Role 1: I am a wrong way driver

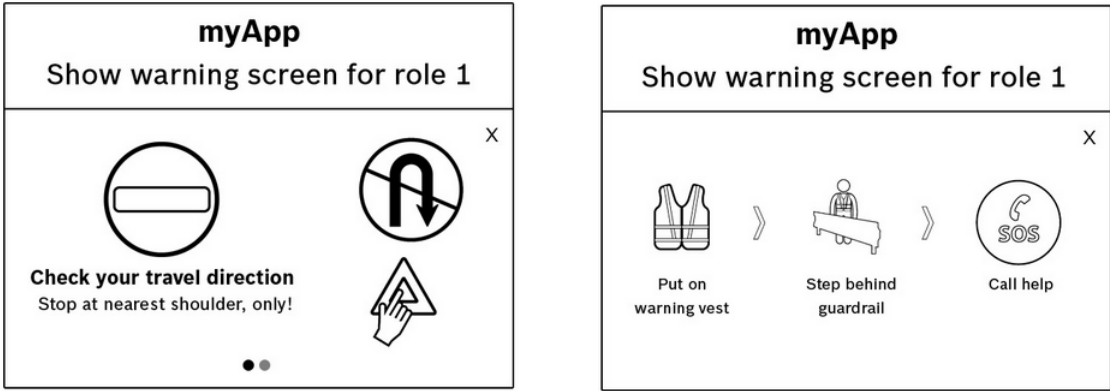


Figure 5: Role 1

#### 2.1.2. Role 2: I am in the vicinity of a wrong way driver

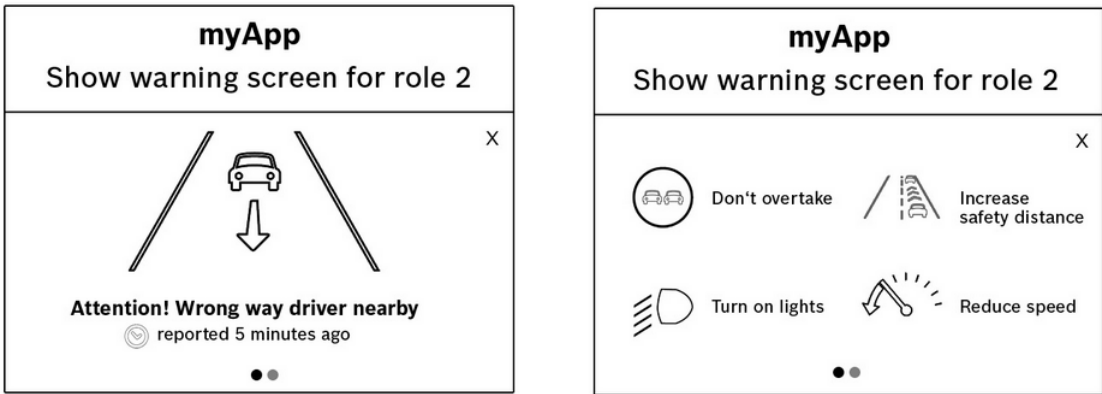


Figure 6: Role 2



## 2.2. Style in General

In general, there are no requirements regarding usage of colours or colour depth. We suggest using no less than 24bit (RGB888) to avoid stepped colour gradients.

## 2.3. Colour & Contrast

In order to improve readability and continuity of design, we recommend to use black/dark backgrounds which are more suitable for in-car usage. It is recommended to use a contrast of about 5:1 between relevant text/icons and the background for night mode and a contrast of about 3:1 for day mode.

## 2.4. Moving Content

No moving content (e.g. video, moving pictures) or complex 3D-objects should be used for representing the WDW warnings.

## 2.5. Icon Style

- ▶ No moving content (e.g. video, moving pictures) or complex 3D-objects should be used for representing the WDW warnings.
- ▶ Automotive guidelines suggest using international/national standards for icons, but for WDW it is only a recommendation.
- ▶ Try not to use metaphors which might be misleading in other countries.
- ▶ It is helpful if important icons are significantly bigger than others (e.g. skip buttons in media apps).
- ▶ Clearly differentiate between icons with button functionality (interaction) and those that only provide information.

## 2.6. Long Text

Long text should be truncated or removed while driving.

## 2.7. Animated Text

No animated/scrolling text at all should be used while driving.

## 2.8. Font Type and Size

Use readable font types and sizes. Font sizes need to be chosen with regard to the distance between driver and device.  
Font Type & Size:

- ▶ Normally, the bigger the better. It is helpful if important text is significantly bigger than the rest.
- ▶ Font styles should be readable at a glance.
- ▶ Text length should be kept as short as possible (keywords).
- ▶ We recommend using strong contrasts.

## 2.9. Samples for Font Type & Style

### 2.9.1. Sample for Font Type and Style Role 1

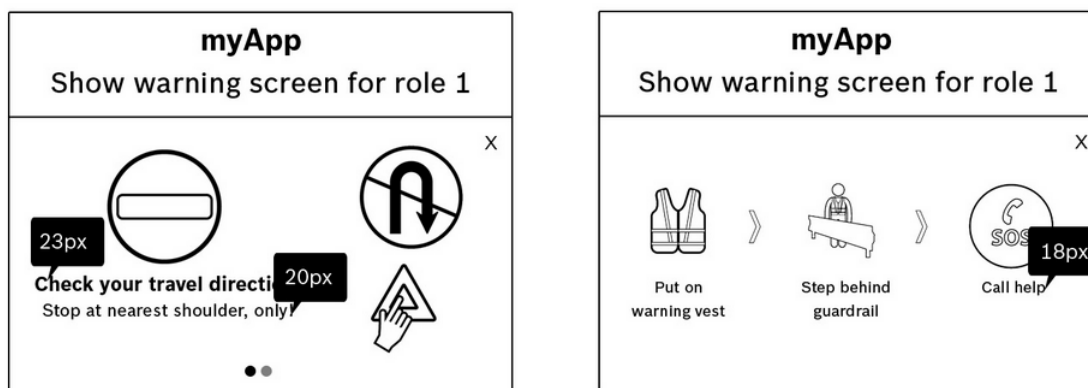


Figure 7: Sample 1

### 2.9.2. Sample for Font Type and Style Role 2

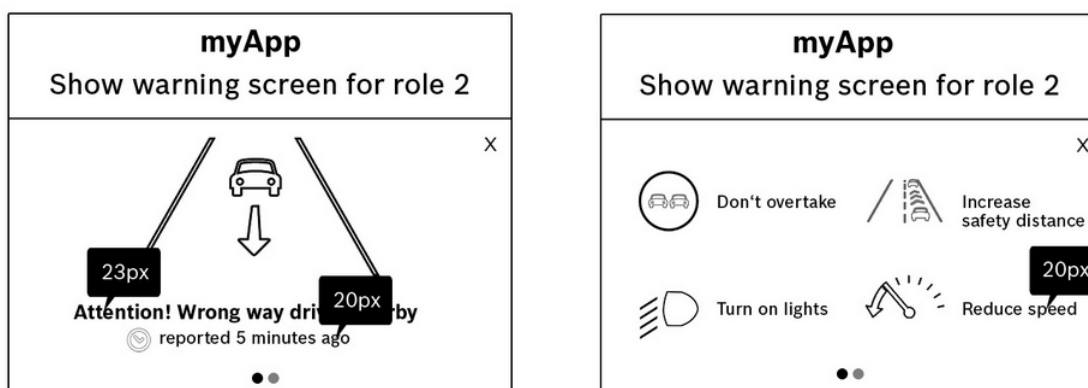


Figure 8: Sample 2

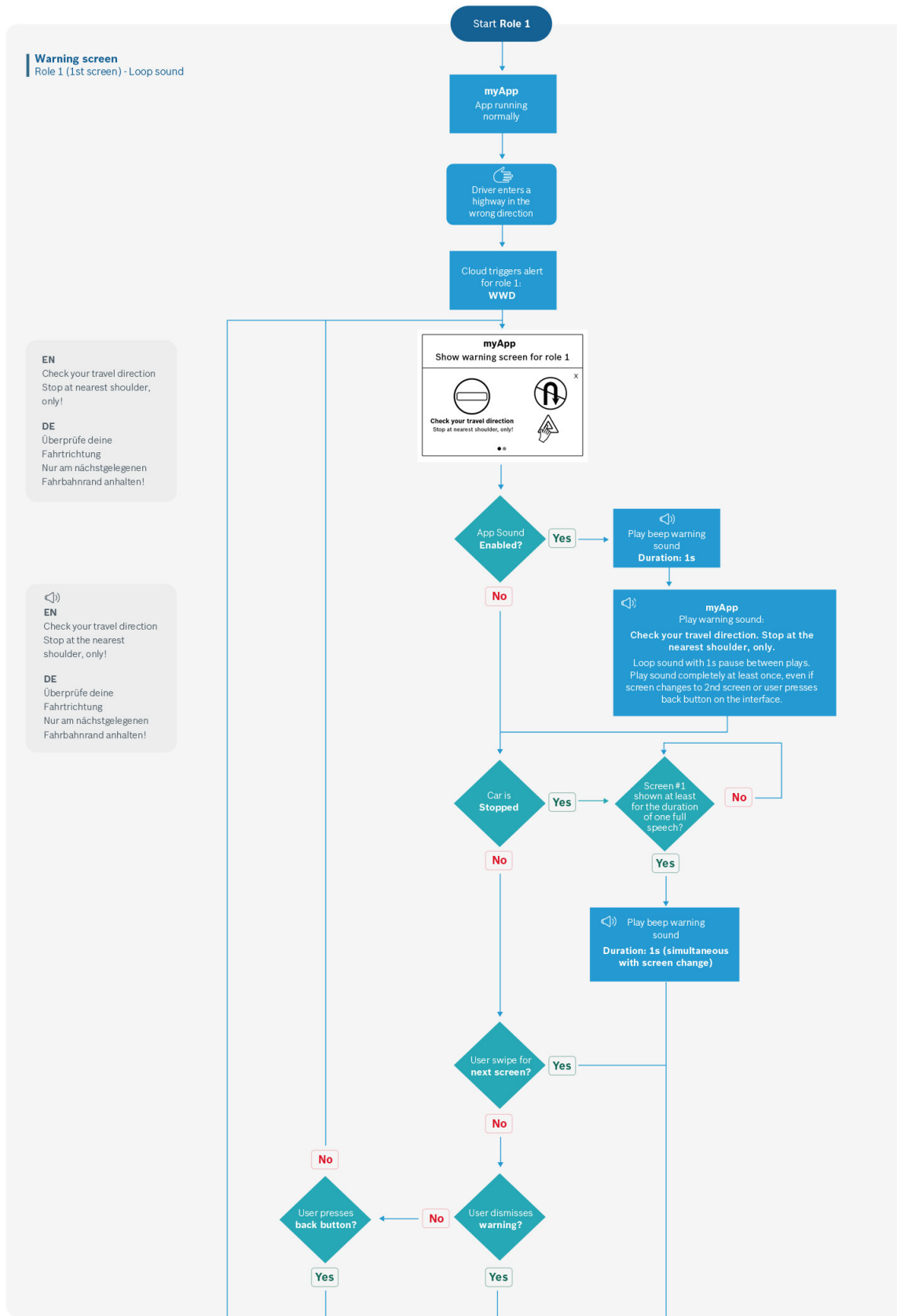
## 2.10. Wording

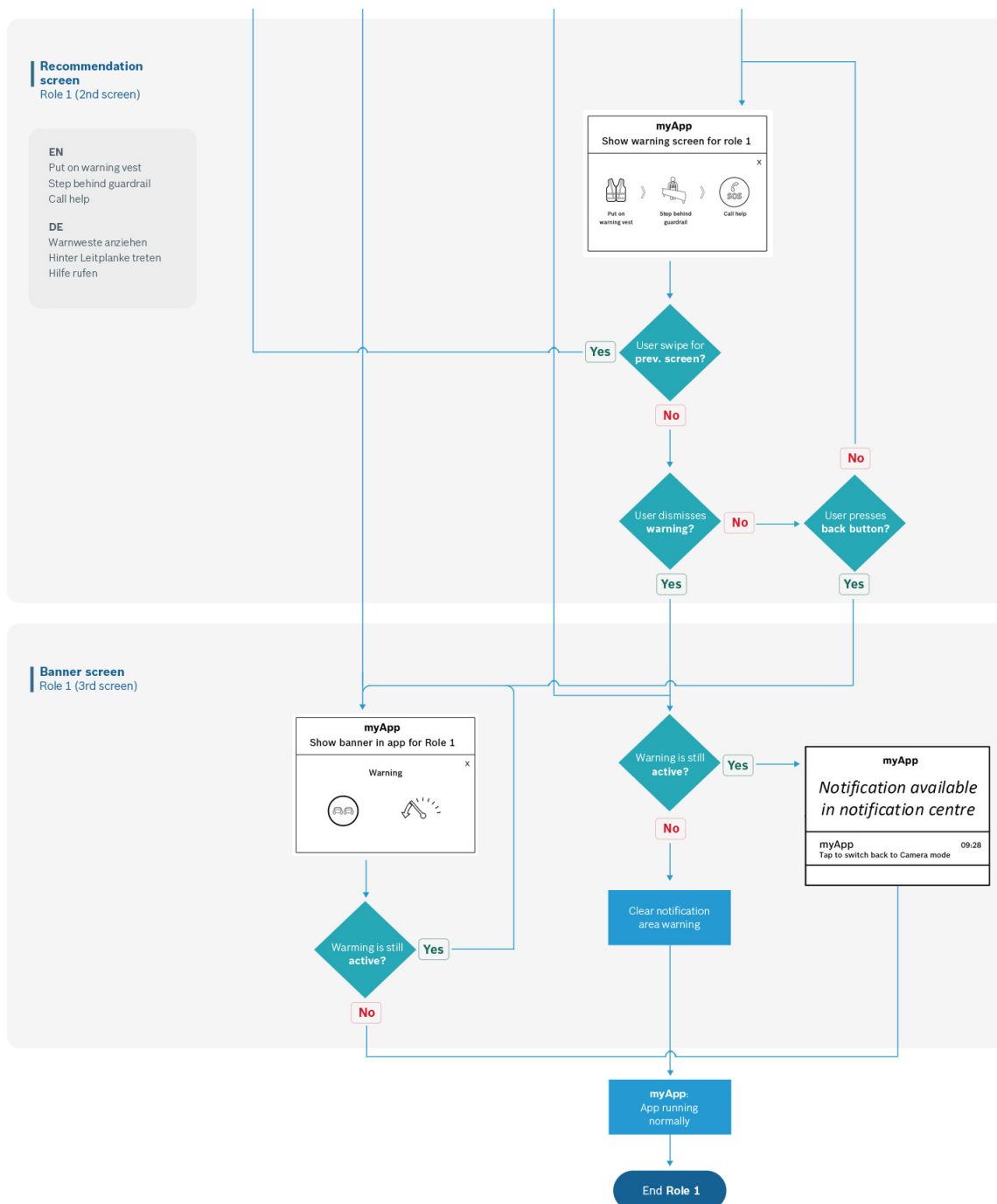
Wording should be concise; don't use unnecessary text. Meaning should be clear at a glance.

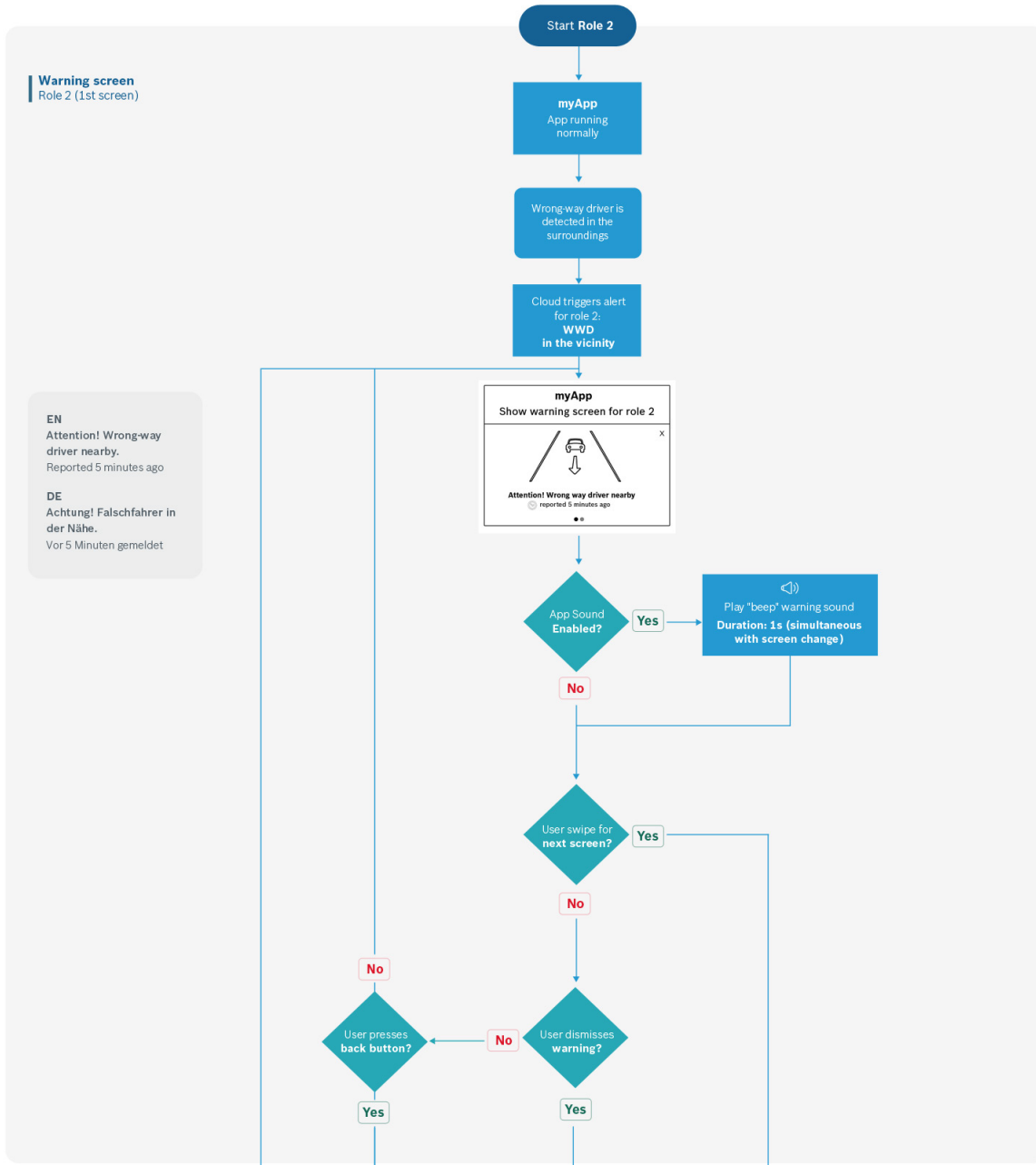
Wording:

- ▶ short and simple
- ▶ acronyms: only if clear (internationally known)
- ▶ abbreviations: only if clear (e.g. min=minutes...)

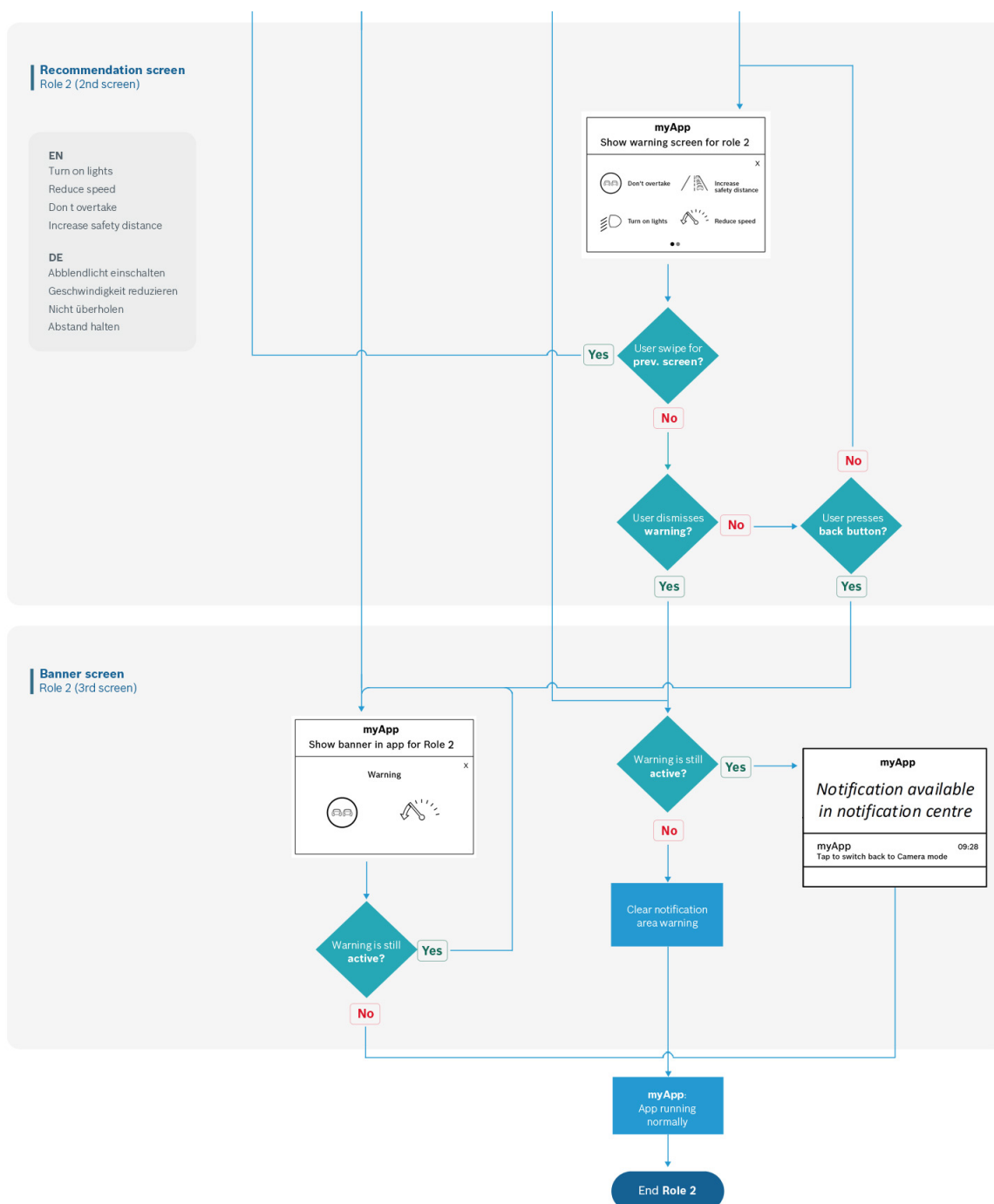
### 3. Flowchart













## Part V. FAQ



## 1. FAQ

### 1.1. Where can I get support?

General support for the WDW-SDK is available here: [support.wdw@de.bosch.com](mailto:support.wdw@de.bosch.com).

### 1.2. How to receive the fleetId?

The fleetId needs to be requested from Bosch.

### 1.3. How can I determine which version of the WDW-SDK I am using?

The version of the SDK is part of its external interface. Use the following code to get the version:

- ▶ Java: `String sdkVersion = WDWClient.VERSION;`
- ▶ Objective-C: `NSString *sdkVersion = [WDWClient version];`

### 1.4. How does the WDW-SDK give relevant data to the integrating application?

A JSON-String is generated and given to the application via callback.



## 2. FAQ - iOS

### 2.1. Which Apple hardware is supported?

The WDW-SDK supports iPhone and iPad devices with iOS 8.0 and higher. The WDW-SDK will not start on devices that have no GPS sensor (e.g. iPad Wi-Fi or iPod Touch).

### 2.2. How to use the integrated push notification solution?

Your application needs to have the Push Notification service enabled based on your *App ID*. The following steps describe the process:

- ▶ Open the App IDs section in your *Apple Developer* account.
- ▶ Find your Application and enable Push Notifications for it.
- ▶ Follow the Apple guide to generate *APNs Certificate* and install it to your OS X Keychain.
- ▶ Export the generated certificate(s) to the p12 format and send to Bosch.
- ▶ Integrate the push service credentials received from Bosch. For further information, see [Integration How To](#).

### 2.3. What size does the WDW-SDK have?

Library File (MB)	Influence on application (MB)
4,0	2,9

### 2.4. Will the WDW-SDK run in background mode?

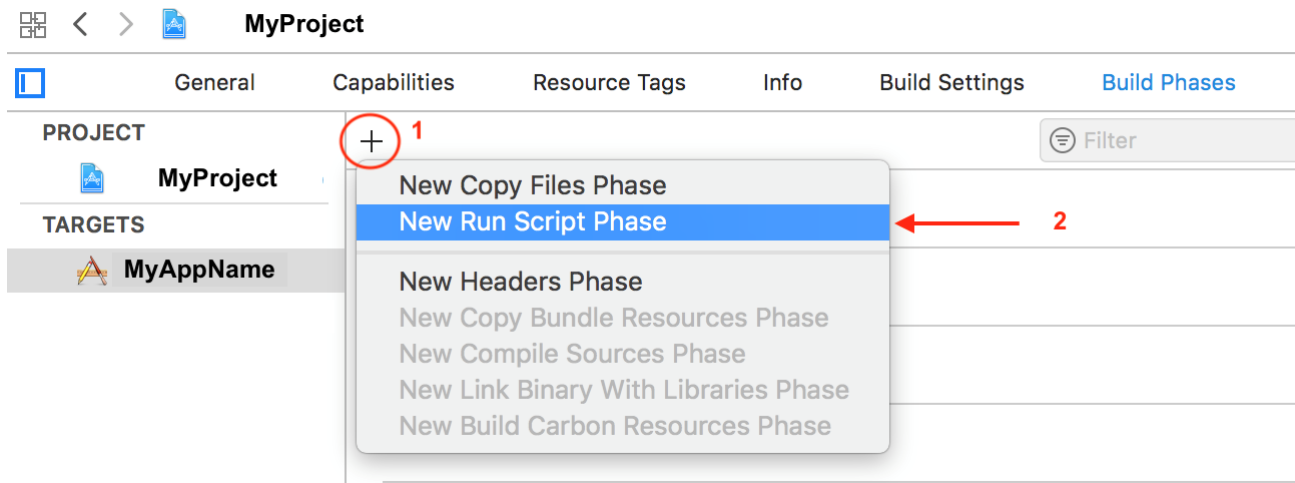
Yes, unless the user has explicitly restricted the Application's location access to foreground mode only. In that case, WDW-SDK will be able to operate only when the Application is active (displayed on the screen).

### 2.5. Does the WDW-SDK support bitcode-enabled applications?

Yes, the WrongWayDriverSDK.framework binary includes the bitcode segments required for distributing bitcode-enabled applications via the Apple App Store.

### 2.6. Does the WDW-SDK support development in iOS simulator?

Yes, the WrongWayDriverSDK.framework contains the binary code for the iOS simulator platform. However, it is necessary to strip out the iOS simulator code when producing the App Store binary. This is achieved by adding a build phase to your Xcode project:



Then, apply the following script:

```
# Remove unused platforms from WrongwayDriverWarningSDK
```

```
WDWSDK_BIN_PATH="${TARGET_BUILD_DIR}/${WRAPPER_NAME}/Frameworks/WrongwayDriverWarningSDK.framework
```

```
REQUIRED_ARCHS=()
```

```
if [ "$ARCHS" == "x86_64" ]; then
    echo "Stripping is not needed"
    exit 0
fi
```

```
if [ ! -f "${WDWSDK_BIN_PATH}" ]; then
    echo "WrongwayDriverWarningSDK is not found at build directory"
    exit 1
fi
```

```
# extract only platforms which MYAPP currently built with
for ARCH in $ARCHS
do
    echo "Processing WrongwayDriverWarningSDK ${ARCH}"
    lipo -extract "${ARCH}" "${WDWSDK_BIN_PATH}" -o "${WDWSDK_BIN_PATH}-${ARCH}"
    REQUIRED_ARCHS+=("${WDWSDK_BIN_PATH}-${ARCH}")
done
```

```
# merge extracted platforms back to 1 file
lipo -o "${WDWSDK_BIN_PATH}-merged" -create "${REQUIRED_ARCHS[@]}"
rm "${WDWSDK_BIN_PATH}"
mv "${WDWSDK_BIN_PATH}-merged" "${WDWSDK_BIN_PATH}"
```

```
# cleanup
rm "${REQUIRED_ARCHS[@]}"
```

```
echo "WrongwayDriverWarningSDK stripped to run only on platforms: ${ARCHS}"
```



## Part VI. Best Practices



## 1. Best Practices - iOS

### 1.1. Detecting GPS hardware availability

In order to maintain good user experience, your application may wish to support devices without GPS sensor (e.g. iPod touch). However, WDW-SDK requires the GPS sensor for its functioning. Thus, the WDW-SDK will report `WDW_Unsupported_Hardware_Error` on attempting to `-startMonitoring` on a device without GPS hardware:

```
@interface MyAppDelegate() <WDWClientDelegate>
...

- (void)wdwClient:(WDWClient *)client didReceiveError:(NSError *)error {
    if (error.code == WDW_Unsupported_Hardware_Error) {
        NSLog(@"WDW-SDK did not start: no GPS sensor available.");
        // notify user if needed
    }
    // handle other errors
}
...
```

Listing 14: WDW-SDK error callback

### 1.2. Dealing with multiple Push Notification providers

Your application might already use Push Notifications. However, the iOS platform provides a single channel for all incoming notifications. Hence, in order to determine whether a push message belongs to the WDW-SDK you need to use the Message Evaluation API as described below:

```
if ([sdkInstance evaluatePushMessage:aPushNotificationPayload] == nil) {
    // <your own message handling code>
}
```

## Part VII.

# Quick Verification Test





## 1. Quick Verification Test

### 1.1. Introduction

The Quick Verification Test (QVT) is a test designed for developers to quickly check and verify the integration of the WDW-SDK in their application. The test cases cover five basic checks to ensure that the WDW-SDK is properly integrated in order to continue developing the application. Each test case will trigger a specific callback method of the WDW-SDK. To verify the test cases, the developer has to observe the callbacks of the WDW-SDK and compare the upcoming events with the given expected results. By setting the log level to `Info`, these events will also appear in the corresponding log output. Refer to the [Integration How To](#) for further details.

### 1.2. Test Setup

1. A smartphone with GPS, accelerometer and gyroscope sensors.
2. An Application with WDW-SDK integrated is installed on the smartphone.

### 1.3. Test Cases

#### 1.3.1. Test Case 1: The SDK starts without any problems

**Preconditions:** 1. Location service is enabled on the smartphone.  
2. All required permissions have been given.

**Test Action:** Create an instance of the SDK.

**Expected Result:** After a few seconds, the SDK will report the availability state `Available`.

**Test Result:**

**Comments:**

#### 1.3.2. Test Case 2: The SDK will not start if GPS is not available

**Precondition:** GPS is unavailable (Location service disabled or Location permission denied).

**Test Action:** 1. Create an instance of the SDK.  
2. Start monitoring.

**Expected Result:** The SDK triggers an error indicating that GPS is unavailable.

**Test Result:**

**Comments:**

#### 1.3.3. Test Case 3: The SDK generates data when entering a highway

**Preconditions:** 1. Location service is enabled on the smartphone.  
2. All required permissions have been given.

**Test Action:** 1. Create an instance of the SDK.  
2. Enter a highway.

**Expected Result:** The SDK starts generating data with a rate of 1Hz.

**Test Result:**

**Comments:**



### 1.3.4. Test Case 4: The SDK does not generate data when not driving on a highway

**Preconditions:**

1. Location service is enabled on the smartphone.
2. All required permissions have been given.

**Test Action:**

1. Create an instance of the SDK.
2. Start monitoring.
3. Drive on a road not in the vicinity of a highway.

**Expected Result:** The SDK doesn't generate any data.

**Test Result:**

**Comments:**

### 1.3.5. Test Case 5: The SDK triggers a wrong way driver warning

**Preconditions:**

1. Location service is enabled on the smartphone.
2. All required permissions have been given.

**Test Action:**

1. Create a custom message evaluator that always returns a warning of type `WDWInFront`.
2. Create a new instance of the SDK using the custom message evaluator.
3. Start monitoring.
4. Request the SDK to evaluate an empty message.

**Expected Result:** The SDK triggers a new warning of type `WDWInFront`.


**Test Result:**

**Comments:** The following code snippet shows an example of an empty message:

```
{
  "meta": {
    "radius": 0,
    "lifetime": 0,
    "headingRelevance": 0
  },
  "data": {
    "driveId": "",
    "uniqueId": "",
    "event": "wrong way event",
    "timeGenerated": "",
    "tags": [
      "driver",
      "data"
    ]
  },
  "location": {
    "gps": {
      "latitude": 0,
      "longitude": 0,
      "altitude": 0,
      "horizontalAccuracy": 0,
      "verticalAccuracy": 0,
      "heading": 0,
      "speed": 0
    }
  }
}
```



## Part VIII. Disclosure Document

 <b>BOSCH</b>	CM OSS Disclosure Document	Date
<b>BSOT</b>	BSOT_WDW_SDK_FlyingDutchman_iOS_3.1.2	10-31-2018

## Open Source Disclosure Document for BSOT\_WDW\_SDK\_FlyingDutchman\_iOS\_3.1.2

This document is provided as part of the fulfillment of OSS license conditions and does not require users to take any action before or while using the product.



 <b>BOSCH</b>	<b>CM OSS Disclosure Document</b>	<b>Date</b>
<b>BSOT</b>	BSOT_WDW_SDK_FlyingDutchman_iOS_3.1.2	10-31-2018

Table of Contents

**1** List of used Open Source Components..... **3**  
**2** Appendix - License Text ..... **4**




 <b>BOSCH</b>	<b>CM OSS Disclosure Document</b>	<b>Date</b>
<b>BSOT</b>	BSOT_WDW_SDK_FlyingDutchman_iOS_3.1.2	10-31-2018

## 1 List of used Open Source Components.

This document contains a list of open source software (OSS) components used within the product under the terms of the respective licenses. The source code corresponding to the open source components is also provided along with the product wherever mandated by the respective OSS license

<b>Name of OSS Component</b>	<b>Version</b>	<b>License</b>	<b>More Information</b>
------------------------------	----------------	----------------	-------------------------

 <b>BOSCH</b>	<b>CM OSS Disclosure Document</b>	<b>Date</b>
<b>BSOT</b>	BSOT_WDW_SDK_FlyingDutchman_iOS_3.1.2	10-31-2018

**2    Appendix - License Text**

Robert Bosch Car Multimedia GmbH. All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights