Contents

Omnis App Framework

Omnis Software Ltd

September 2019

User Guide

Setup – Android

Instructions for setting-up the Omnis App Framework on Android devices.

Support Library

The Omnis Interface uses the AndroidX support library. At time of publication, this is a relatively new replacement to the previous Support Library used by Android apps, so you may need to migrate existing code to AndroidX.

If you are creating a new project, there is currently a checkbox shown in the New Project wizard - "**Use androidx.* artifacts**" - make sure to enable this.

Add The Library To Your Project

- Download and unzip the omnisInterface library.
- From your Android Studio project, right-click your app module, and select Open **Module settings** from the context menu.
- Add a new module, and select **Import Gradle Project** from the wizard which opens, then select the expanded **omnisInterface** folder.

- Because the omnisInterface library is installed locally, rather than fetched from a repository by Gradle, you will need to include its dependencies when you build.
  - Add the following to your app's **build.gradle** file's **dependencies** section:

```
// Dependencies required by OmnisInterface (as we're not pulling it in from a repository):
implementation omnisInterfaceImplementationDependencies.values()
implementation omnisInterfaceAPIDependencies.values()
```

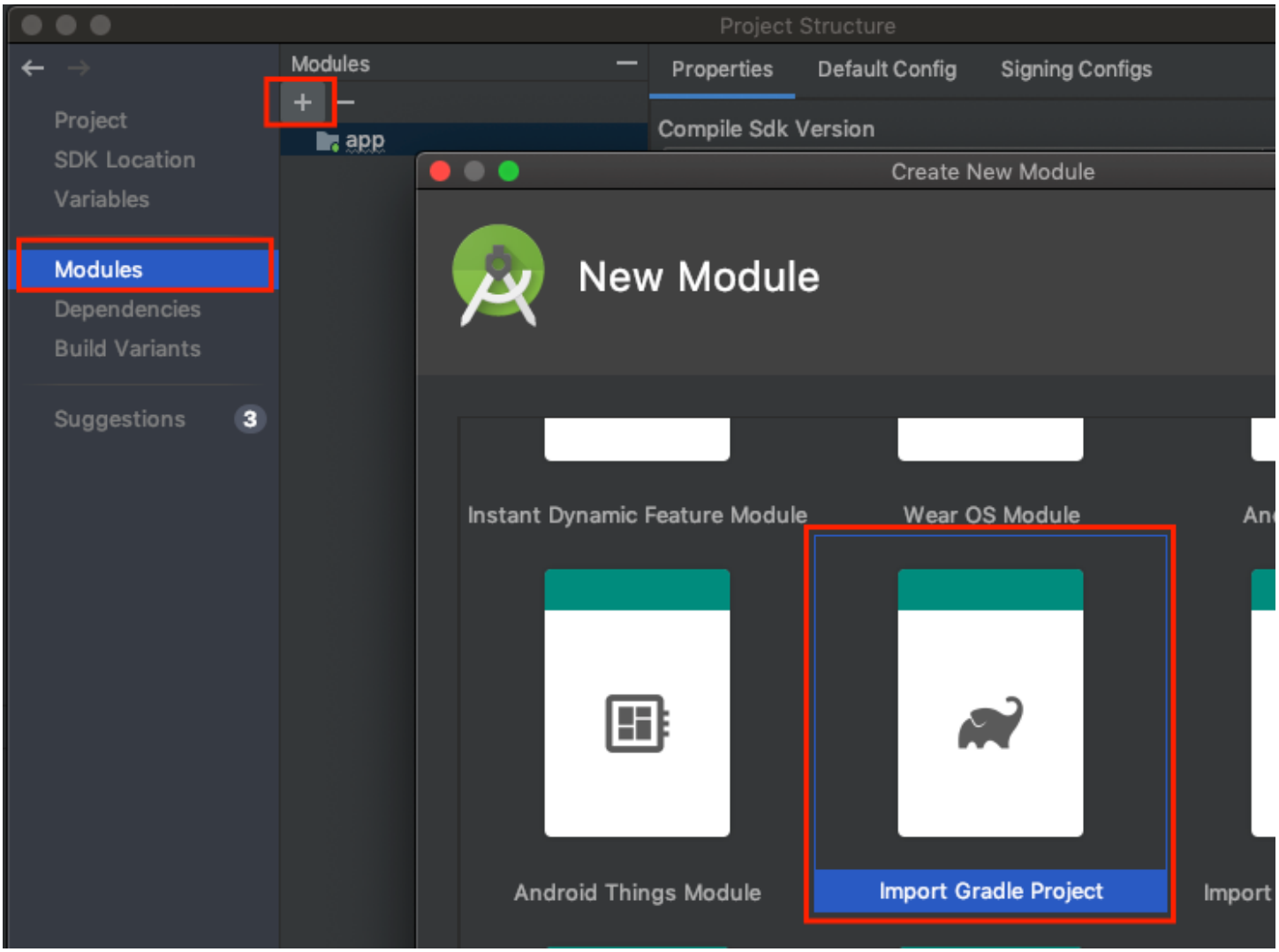
Project Structure
Modules
Properties
Default Config
Signing Configs
Project
SDK Location
Variables
Modules
Dependencies
Build Variants
Suggestions
3
app
Compile Sdk Version
Create New Module
New Module
Instant Dynamic Feature Module
Wear OS Module
Android Things Module
Import Gradle Project

Figure 1:

And add the following to the top of that file:

```
apply from: "../omnisInterface/omnisinterface_dependencies.gradle"
```

Add the omnisInterface module as a **module dependency** of your app (again accessed via the **module settings** window):

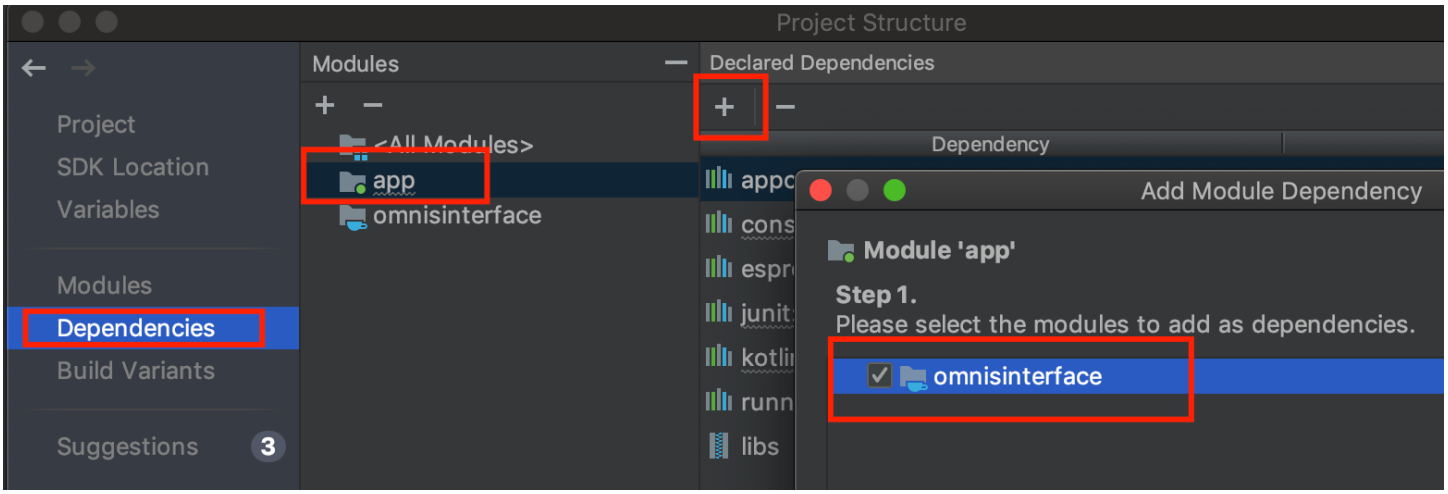

Figure 2:

- In order to update the omnisInterface in the future, just replace the **omnisInterface.aar** and **omnisInterface_dependencies.gradle**.

Instantiate The Omnis Interface

- Add an **OMWebContainer** view to your layout - this view will become the container which hosts the webview to run your Remote Forms etc.

Figure 3:

- Construct an OmnisInterface instance, passing your OMWebContainer to the constructor.
  - Make sure you keep this OmnisInterface in scope.

Example

```
private lateinit var mOmnisInterface: OmnisInterface
override fun onStart() {
  super.onStart()
  if (!this::mOmnisInterface.isInitialized) {
    val webContainer = findViewById<OMWebContainer>(R.id.omweb_container)
    mOmnisInterface = OmnisInterface(webContainer, this)
  }
}
```

Setup - iOS

Instructions for setting-up the Omnis App Framework on iOS devices.

Add the Framework

Add the framework to your project's **Embedded Binaries**:

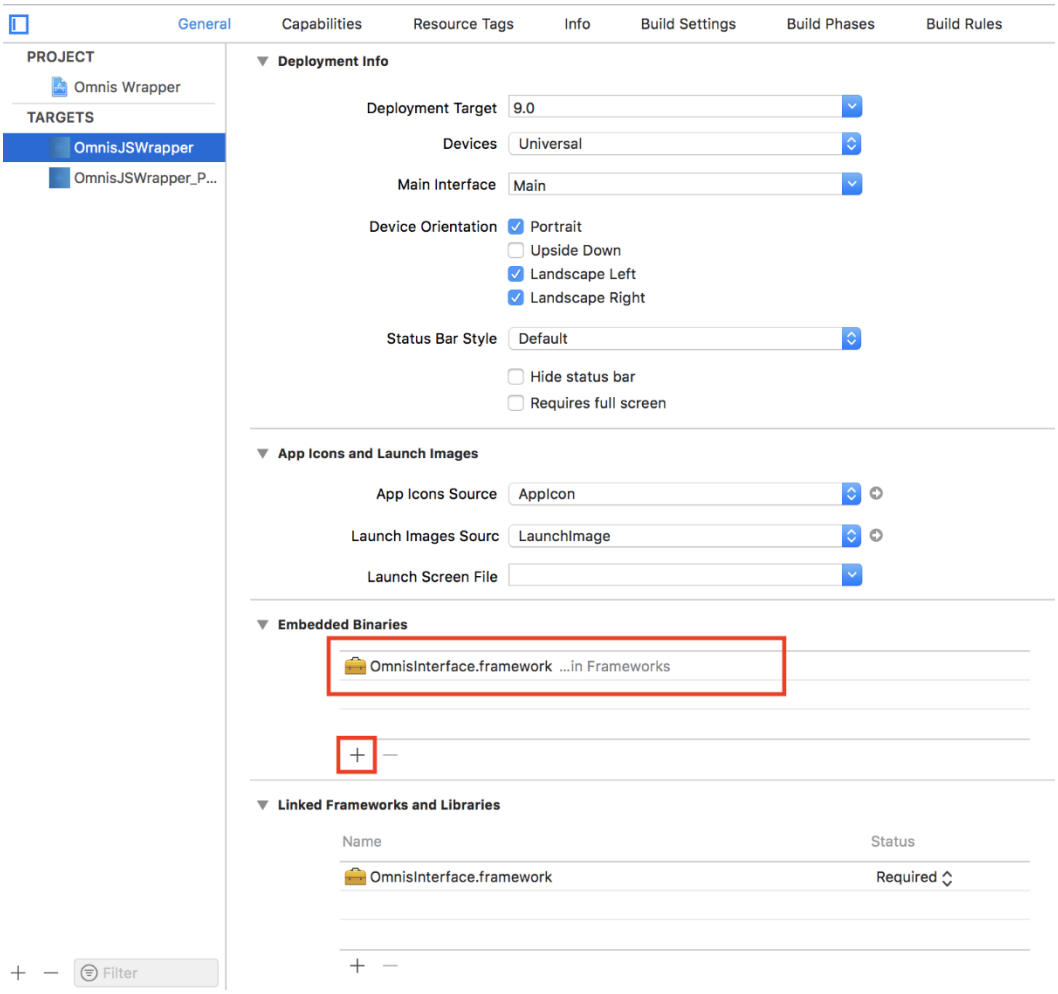

Figure 4:

Create a UIView, and set its Class to **OMWebContainer** - this view will become the container which hosts the webview to run your Remote Forms etc.

- Construct an OmnisInterface instance, passing your OMWebContainer to the constructor.
  - Make sure you keep this OmnisInterface in scope.

Example

```
class ViewController: UIViewController {
  var omnisInterface: OmnisInterface!
@IBOutlet weak var webviewContainer: OMWebContainer!
  ...
  override func viewDidLoad() {
    super.viewDidLoad()
    omnisInterface = OmnisInterface(webviewContainer: webviewContainer, viewController: self)
  }
  ...
}
```

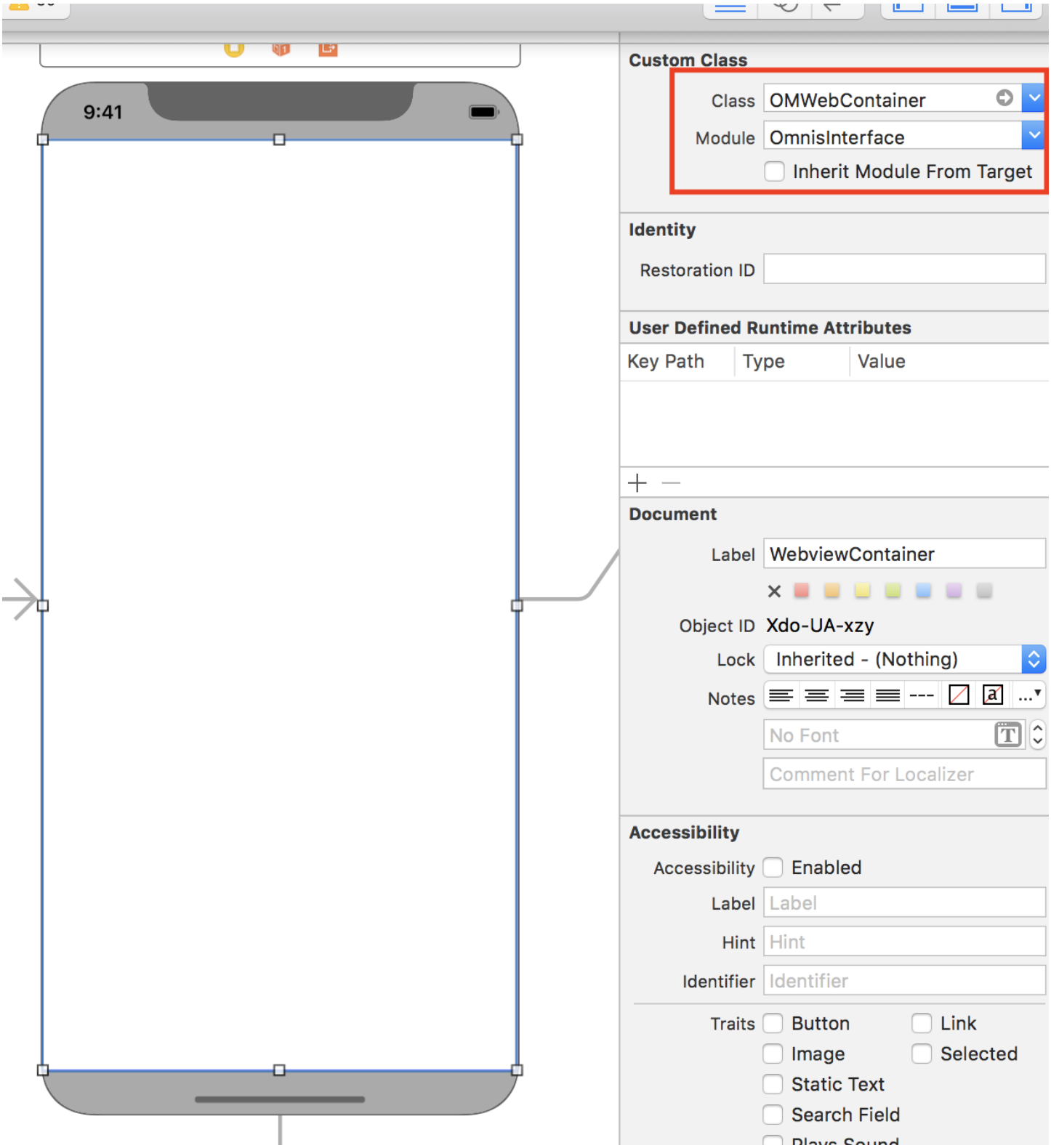
9:41
Custom Class
Class
OMWebContainer
Module
OmnisInterface
Inherit Module From Target
Identity
Restoration ID
User Defined Runtime Attributes
Key Path
Type
Value
Document
Label
WebviewContainer
Object ID
Xdo-UA-xzy
Lock
Inherited - (Nothing)
Notes
No Font
Comment For Localizer
Accessibility
Accessibility
Enabled
Label
Label
Hint
Hint
Identifier
Identifier
Traits
Button
Link
Image
Selected
Static Text
Search Field

Figure 5:

Stripping The Framework

The OmnisInterface Framework is provided as a *Fat Universal binary*, which contains slices for both device & simulator architectures.

To keep the size of you built app down, and to allow your app to be submitted to the App Store, you should strip unused architectures (i.e. Simulator architectures from device builds).

You can do this by:

- In your Project's settings for each **target**:
- Under **Build Phases**, add a **Run Script** section after the **Embed Frameworks** section, with the following content:

```
# Uncomment below to log output:
#exec > /tmp/${PROJECT_NAME}_striparchs.log 2>&1
APP_PATH="${TARGET_BUILD_DIR}/${WRAPPER_NAME}"
# This script loops through the frameworks embedded in the application and
# removes unused architectures.
# Thanks to: http://ikennd.ac/blog/2015/02/stripping-unwanted-architectures-from-dynamic-libraries-in-xcode/
find "$APP_PATH" -name '*.framework' -type d | while read -r FRAMEWORK
do
  FRAMEWORK_EXECUTABLE_NAME=$(defaults read "$FRAMEWORK/Info.plist" CFBundleExecutable)
  FRAMEWORK_EXECUTABLE_PATH="$FRAMEWORK/$FRAMEWORK_EXECUTABLE_NAME"
  echo "Executable is $FRAMEWORK_EXECUTABLE_PATH"
  EXTRACTED_ARCHS=()
  for ARCH in $ARCHS
  do
    echo "Extracting $ARCH from $FRAMEWORK_EXECUTABLE_NAME"lipo -extract "$ARCH" "$FRAMEWORK_EXECUTABLE_PATH"
    EXTRACTED_ARCHS+=("$FRAMEWORK_EXECUTABLE_PATH-$ARCH")
  done
  echo "Merging extracted architectures: ${ARCHS}"
  lipo -o "$FRAMEWORK_EXECUTABLE_PATH-merged" -create "${EXTRACTED_ARCHS[@]}"
  rm "${EXTRACTED_ARCHS[@]}"
  echo "Replacing original executable with thinned version"
  rm "$FRAMEWORK_EXECUTABLE_PATH"
  mv "$FRAMEWORK_EXECUTABLE_PATH-merged" "$FRAMEWORK_EXECUTABLE_PATH"
done
```

Loading Forms

Once you have set up your OmnisInterface, you can call methods on it (or the objects it exposes as properties - such as scafHandler) to interact with the JS Client.

To load a form, simply call OmnisInterface's loadURL method, giving it the full URL to your Remote Form's .htm file.

Android Example

```
val theUrl = URL("https://mysite.com/myForm.htm")
mOmnisInterface.loadURL(theUrl)
```

iOS Example

```
let theUrl = URL(string: "https://mysite.com/myForm.htm")
self.omnisInterface.loadURL(url: theUrl!)
```

Loading Progress Events

In order to be notified when a form begins/ends loading etc, you need to assign a class which implements theOMWebNavigationDelegate protocol to your OmnisInterface's webNavigationDelegate property.

The delegate methods will then be called at the appropriate times.

Offline Forms

You must initialise your OmnisInterface's scafHandler by calling initScafController, in order to enable support for offline forms.

This sets the ScafHandler up with details of your Omnis server, and offline form details etc.

Once this has been done, and SCAFs have been updated/extracted*, you can load the configured offline form by calling:

```
self.omnisInterface.scafHandler.loadOfflineForm()
```

* Note that this is an asynchronous process, so you need some way of determining when the offline form is ready:

Implement Delegate

You need to assign some class, which implements the OMScafHandlerDelegate to the scafHandler's delegate property.

The delegate's onOfflineModeConfigured method will then be called when your offline form is ready (either after startup, once everything is prepared, or after you call updateScafs).

Custom Messaging - iOS

To implement custom messaging between the JS Client and your app:

Omnis to iOS

- In your iOS app, register a CommsDelegate against the OmnisInterface.
- Call a **form**'s **sendWrapperMessage()** method from JavaScript running in the JS client.
  NOTE: This method did not exist before Omnis Studio **8.1.7**. If you are using a prior version, load the following JavaScript **after omjsclnt.js**:

```
omnis_form.prototype.sendWrapperMessage = function(id, data, formCallbackMethod)
{
if (formCallbackMethod)
{
  var callbackString = 'var form = jOmnis.getOmnisForm(jOmnis.instIndexFromIdent(' + this.ident +
    '), jOmnis.formIndexFromIdent(' + this.ident + '));' +
    '[].splice.call(arguments, 0, 0, "' + formCallbackMethod + '");' + // Insert the method name as the first
    'return form ? form.callMethod.apply(form, arguments) : null;';
}
else
  callbackString = "";
sendDeviceRequest(id, callbackString, data);
};
```

- Pass this function the following parameters:
  - an **ID** to identify your message
  - any **data** (should be a JSON object)
  - optionally the name of a **form method** to call with the results.

Example

```
JavaScript:__form.sendWrapperMessage("myMessage", {firstName: "Alan", lastName: "Alphabet"}, "$messageReturn")
```

- Your CommsDelegate's messageFromJSClient(data:) method will be called, with the JSON payload you passed in, received as a Dictionary.

- The dictionary should contain the following members:
  - **ID**: The ID you sent for your message.
  - **Data**: A JSON dictionary containing the data you sent.
  - **retID**: (Optional) JavaScript function code text, to be executed to pass the results back to the JS client.
- Return true if default handling should occur, else return false if you've handled the message.

iOS to Omnis

- Call omnisInterface's callJS method to execute whatever JS you wish.
  - You can optionally pass a completion handler to be called with any results from the JS call.

Or, in order to call a callback function passed to the wrapper from **sendWrapperMessage** above, you can use your OmnisInterface's callJSFunctionText method, passing it the value given in messageFromJSClient's **retID** member as the functionText.

Example

```
extension MainViewController: OMCommsDelegate
{
  func messageFromJSClient(data: [String : AnyObject]?) -> Bool! {
  if let id = data?["ID"] as? String
  {
   switch id
      {
        case "myMessage":
        let firstName = data?["Data"]?["firstName"] as? String ?? ""
        let lastName = data?["Data"]?["lastName"] as? String ?? ""
        // TODO: Something with the data
        // Call back to an Omnis method, if one was provided:
        if let returnFunc = data?["retID"] as? String
        {
          omnisInterface.callJSFunctionText(returnFunc, params: [
            "'\(id)'", // Will be first param passed to method
            "'\(firstName)'", // Will be 2nd param passed to method
            "'\(lastName)'" // Will be 3rd param passed to method
            ], completionHandler: nil)
        }
        break;
        default:
          return false; // Let default handling occur for messages we're not explicitly handling.
      }
    }
    print("No ID provided!")
    return false // Let default handling occur
  }
}
```

If you followed the examples above, after sending the message to the wrapper, the form's **$messageReturn** method will be called, with the following parameters:

p1: "myMessage" (The message ID)

p2: "Alan"

p3: "Alphabet"

Photo Functionality - Android

In order for the photo functionality to work with your app, you must create a File Provider, which allows sharing of files within your app's internal storage area (So the Camera app can save files there).

1) Create a resource file to define the File Provider:

Create an **xml** resource with content to provision a folder within the "**files-path**" area as shareable.

Example - fileprovider.xml

```
<?xml version="1.0" encoding="utf-8"?>
<paths>
  <files-path name="photos" path="photos" />
</paths>
```

By default, it should have a path named "photos".

2) Include the File Provider in your Manifest

Edit your **AndroidManifest.xml** file, and add the File Provider to the **Application** section:

Example

```
<provider
  android:name="android.support.v4.content.FileProvider"
  android:authorities="net.omnis.fileprovider"
  android:exported="false"
  android:grantUriPermissions="true">
  <meta-data
    android:name="android.support.FILE_PROVIDER_PATHS"
    android:resource="@xml/fileprovider" />
</provider>
```

Set the **android:authorities** attribute to your own domain identifier.

3) Override File Provider defaults if necessary

The **OMImages** class has the following public static members, which should be overridden with your own values if they differ from the defaults:

FILE_PROVIDER_AUTHORITY: Should match your **android:authorities** above (default = "net.omnis.fileprovider").

FILE_PROVIDER_PATH: Should be the **path** specified for your **files-path** entry in fileprovider.xml (default= "photos").

Barcode Functionality - Android

Barcode functionality on Android makes use of Google's ML Kit framework, and as such requires a Firebase project.

You need to create a project in Firebase, then download the **Google-Services.json** file for your project, and add to the root of your App module in your Android Studio project.

API Reference ==========

OmnisInterface

The main interface between your native app and Omnis.

Statics

Property Name	Type	Description
DEVICE_ID	String	A read-only unique identifier for this device. May be reset if the app is uninstalled then later re-

Properties

Property Name	Type	Description
features	Features	Contains the device features this app supports.
settings	settings	The current settings used by the interface.
database	LocalDBController	
scafHandler	ScafHandler	Controls access to and features of offline forms.
viewController	UIViewController	The View Controller the OmnisInterface was initialised
webNavigationDelegate	OMWebNavigationDelegate	The delegate which receives page navigation events fr
commsDelegate	OMCommsDelegate	The delegate which receives messages from the JS Clie

Methods

Method Name	Description
init(webviewContainer, [viewController, settings])	Constructor: Initialises the interface, and creates a webview within the cont
loadURL(url, [withParams])	Loads the passed URL in the attached webview.
callJS(jsString, [completionHandler])	Executes the passed JS string in the webview. Optional callback with result
callJSFunctionText(functionText, [params, completionHandler])	Executes the passed JS function text in the webview (inside an IIFE). Option callback with result.

init()

init(webviewContainer, [viewController, settings])

Constructor: Initialise the OmnisInterface, ready to be used.
This must be called before you can interact with the OmnisInterface.

Parameters

Name	Type	Description
webviewContainer	OMWebContainer!	A UIView with class OMWebContainer, the Omnis JS Client. It must have no ch
viewController	**(Optional)** UIViewController?	The viewController on which the Omni views (e.g. for barcode scanning, dialog will attempt to find the webviewContai
settings	**(Optional)** Settings	Initial Settings to use with the OmnisIn Settings will be created during initializa

Returns

None.

loadURL()

loadURL(url, [withParams])

Loads the passed URL in the attached webview.
Use this to load your Remote Form.

Parameters

Name	Type	Description
url	URL	The URL to load.
withParams	Dictionary	A Dictionary of String names and values to set as URL Query parameters. These will be received as a JSON string, in the form's $construct row (first parameter to $construct - make it of type Row)

Returns

None.

callJS()

callJS(jsString, [completionHandler])

Executes the passed JS code in the webview. Calling an optional completion handler once complete.
If called from a background thread, this will be posted to run on the main thread as soon as possible.

Parameters

Name	Type	Description
jsString	String!	The JS code to execute.
completionHandler	**(Optional)**func (Any?, Error?) -> Void	A handler to be called with the results of the ex

Returns

None.

Example

```
omnisInterface.callJS("12 + 7", completionHandler: {(result, error) in
    print(result!)
})
```

callJSFunctionText()

callJSFunctionText(functionText, [params, completionHandler])

Executes the passed JS function text in the webview (inside an IIFE). Calling an optional completion handler once complete.
If called from a background thread, this will be posted to run on the main thread as soon as possible.
Used to call a callback function provided with a device message request.

Parameters

Name	Type	Description
functionText	String	The JS Function content to run.
params	[String]	An array of params, expressed as strings, to be used as ra must be surrounded in extra (escaped) quotes.
completionHandler	(Any?, Error?) -> Void	A handler to be called with the results of the execution.

Returns

None.

Features

A class containing the device features supported by this app.

Statics

Name	Type
FeatureTypes	struct

FeatureTypes

Contains constant values for the built-in features.

Key	Description
NATIVE_DIALOGS	The app supports the overriding of Omnis' JS dialogs with native dialogs. (It handles Comms delegate. The OmnisInterface does by default)
SHOW_PDF	The app supports displaying PDFs. (It handles the "**showPDF**" message sent to the C does by default)
PRINT_PDF	The app supports printing of PDFs. (It handles the "**printPDF**" message sent to the C does by default)
STORED_PREFS	The app supports a native implementation for storing preferences, rather than using "**savePref**" & "**loadPref**" messages sent to the Comms delegate. The OmnisInterface

Methods

Method Name	Description
addFeature(featureID)	Add a feature to the list of supported features.
removeFeature(featureID)	Remove a feature from the list of supported features.
getFeatures()	Get an array of the features (by ID) which are currently suppo
hasFeature(featureID)	Checks whether a feature has been enabled.

addFeature()

addFeature(featureID)

Add a feature to the list of supported features.

Parameters

Name	Type	Description
featureID	String	A string denoting the feature to add**.** May be a custom String, or a FeatureTypes value.

Returns

None.

removeFeature()

removeFeature(featureID)

Remove a feature from the list of supported features.

Parameters

Name	Type	Description
featureID	String	A string denoting the feature to remove. May be a custom String, or a FeatureTypes value.

Returns

None.

getFeatures()

getFeatures()

Get an array of the features (by ID) which are currently supported.

Parameters

None.

Returns

Type	Description
[String]	An array of featureID strings.

hasFeature()

hasFeature(featureID)

Checks whether a feature has been enabled.

Parameters

Name	Type	Description
featureID	String	The ID of the feature you wish to check. May be a custom String, or a FeatureTypes value.

Returns

Type	Description
Bool	Whether the feature is enabled.

Settings

The current settings used by the OmnisInterface.

Change these settings to alter various aspects of behaviour.

These settings are not saved to disk.

Statics

Name	Type	Description
SettingNames	struct	Contains the setting key names for built-in settings.

SettingNames

Contains the setting key names for built-in settings.

Key	Description
USE_LOCAL_TIME	If true, when running in offline mode, dates will not be converted to UTC during transmission (online mode reads from remote task's $localtime property).
EXPECT_SCAFS	If true, give an error if SCAF files are not found initially. Must be set before calling initScafController().

Methods

Method Name	Description
setSetting(name, value)	Sets the value for a named setting.
setSettings(dictionary)	Sets multiple settings from a provided Dictionary.
getSetting(name, defaultValue)	Gets the current value for a named setting.
resetSettings()	Resets the settings to their default initial values.

setSetting()

setSetting(name, value)

Sets the value for a named setting.

Parameters

Name	Type	Description
name	String!	The name of the setting to apply. Could be a SettingNames value, or a custom string. Overwrites any existing setting with the same name.
value	Any	The value for the setting.

Returns

None.

setSettings()

setSettings(dictionary)

Sets multiple settings from a provided Dictionary.

Parameters

Name	Type	Description
settings	Dictionary	A Dictionary of key-values. The keys should refer to setting names (e.g. SettingNames values)

Returns

None.

getSetting()

getSetting(name, defaultValue)

Gets the current value for a named setting.

Parameters

Name	Type	Description
name	String!	The name of the setting whose value should be returned.
defaultValue	Any?	A default value to return in the event that the setting was not found.

Returns

Type	Description
Any?	The value for the specified setting (or the default value).

resetSettings()

resetSettings()

Resets the settings to their default initial values, i.e. any custom settings will be removed, and the built-in settings will be reverted to their initial values.

Parameters

None.

Returns

None.

LocalDBController

An interface into the local database used by Omnis' $sqlobject.

Must be initialised for local database and Sync Server support to work.

Properties

None

Methods

Method Name	Description
initLocalDatabase([dbName])	Initialises local Database and Sync Server su
sendLocalDBRequest(request)	Adds a database request to the queue. Only "ExecuteSQL" message.
close()	Closes the local database resources (threads

initLocalDatabase()

initLocalDatabase([dbName])

Initialises local Database and Sync Server support. This must be called in order for the JS Client's local database support to work (and the associated request spooling thread, etc).

Parameters

Name	Type	Description
dbName	String	The name (including ".db" extension) for the local SQLite database to be used. If not provided, "local.db" wi

Returns

None.

sendLocalDBRequest()

sendLocalDBRequest(request)

Adds a database request to the queue. Only for use with the payload sent to the Comms delegate's "ExecuteSQL" message. Once executed, the results will be sent back to the JS Client's omnis_sql_callbackFromWrapper (internal) method. So this is not suitable for making general SQL requests from outside the JS Client.

Parameters

Name	Type	Description
request	String!	A JSON string describing the database request.

Returns

None.

close()

close()

Closes the local database resources (threads etc).

Parameters

None.

Returns

None.

OMCommsDelegate

Contains methods which will be called when communication messages from the JS Client are received.

Accessed via OmnisInterface's commsDelegate property.

Methods

Method Name	Description
messageFromJSClient(data, omnisInterface)	Called whenever a m

messageFromJSClient

messageFromJSClient(data, omnisInterface)

Called whenever a message to the app comes in from the JS Client. It should return a boolean denoting whether the default handling (if any) should occur for this message.

Parameters

Name	Type	Description
data	[String: AnyObject]	The JSON data passed with the message. For built-in messages, th (String) member, a “data” member (usually [String: AnyObject]), an function to call to pass the results back to the JS Client)
omnisInterface	OmnisInterface	The OmnisInterface which is calling this delegate.

Returns

Type	Description
Bool	True if default handling for this message should occur. False if you’ve handled it yourself and do not want default behaviour.

iOS Example

```
extension MyClass: OMCommsDelegate
{
  func messageFromJSClient(data: [String:AnyObject], omnisInterface: OmnisInterface) -> Bool!
  {
    let ID = data["ID"] as? String // The ID for the message action.
    switch ID
    {
        case "myMessage"?:
          // Handle a custom message
          let theMessage = data["myKey"] as? String ?? ""
          handleMyMessage(theMessage)
          return true // We've handled this message
        default:
          return false // We've not handled the message - allow the default handling to try.
    }
  }
}
```

OMWebNavigationDelegate

Provides callbacks for when web navigation events occur (a page starts/end loading etc).

Accessed via OmnisInterface’s webNavigationDelegate property.

Methods

Method Name	Description
omnisBeginLoading(webView, navigation)	Called when t
omnisLoadingComplete(webView, navigation)	Called when t
omnisLoadingFailed(webView, navigation, error)	Called when t

omnisBeginLoading()

```
omnisBeginLoading(webView, navigation)
```

Called when the webview begins loading a page.

Parameters

Name	Type	Description
webView	WKWebView	The webview
navigation	WKNavigation	The navigation object which initiated the page load.

Returns

Type	Description
Bool	True if you have handled the event, or false if you wish default handling to occur. Default handling is to add a loading overlay to

omnisLoadingComplete()

```
omnisLoadingComplete(webView, navigation)
```

Called when the webview successfully completes loading of a page.

Parameters

Name	Type	Description
webView	WKWebView	The webview.
navigation	WKNavigation	The navigation object which initiated the page load.

Returns

Type	Description
Bool	True if you have handled the event, or false if you wish default handling to occur. Default handling is to remove any loading overlay from the webView.

omnisLoadingFailed()

```
omnisLoadingFailed(webView, navigation, error)
```

Called when the webview fails to load a page.

Parameters

Name	Type	Description
webView	WKWebView	The webview.
navigation	WKNavigation	The navigation object which initiated the page load.
error	Error	The error that occurred.

Returns

Type	Description
Bool	True if you have handled the event, or false if you wish default handling to occur. Default handling is to remove any loading overlay from the webView & show an OK message with the error.

ScafHandler

A class used to administer the handling of SCAFs (Serverless Client Application Files), and functionality to do with running offline forms.

The only instance of this you will need will be accessed via your OmnisInterface's scafHandler property.

Properties

Property Name	Type	Description
delegate	OMScafHandlerDelegate	A delegate to receive callb

Static Methods

Method Name	Description
deleteOfflineFiles(atSubfolder)	Deletes any offline fi
moveOfflineFiles(fromSubfolder, toSubfolder)	Moves all of the offlir

deleteOfflineFiles()

deleteOfflineFiles(atFolder)deleteOfflineFiles(atFolder, context)

Deletes the specified folder (relative to the app's Documents directory)

Parameters

Name	Type	Description
atSubfolder	String	A relative path to the folder to delete. E.g. "subfolder/moreFiles"
context	Context	The Context from which to resolve the file paths.

Returns

Type	Description
Bool	Whether the directory was successfully found & deleted.

moveOfflineFiles()

moveOfflineFiles(fromSubfolder, toSubfolder) throwsmoveOfflineFiles(fromSubfolder, toSubfolder, context) throws

Moves the contents of a subfolder (relative to Documents directory) to another subfolder.
The destination folder must not exist.
Throws an Error on failure.

Parameters

Name	Type	Description
fromSubfolder	String	A relative path to the folder to delete. E.g. “subfolder/moreFiles”
toSubfolder	String	A relative path to the folder to move to. Must not exist.
context	Context	The Context from which to resolve the file paths.

Returns

None.

Methods

Method Name	Description
initScafController(inSubfolder, formName, scafName, omnisWebUrl, [omnisServer, omnisPlugin])	Initialise the SCAF controller, to enable offline mode. This must be called before oth (non-static) methods.
updateScafs()	Query the configured Omnis server for any updates. Any available updates will be downloaded and applied.
loadOfflineForm([queryParams])	Load the configured offline form.

initScafController()

initScafController(inSubfolder, formName, scafName, omnisWebUrl, [omnisServer, omnisPlugin])

Initialise the SCAF controller, to enable offline mode. This must be called before other (non-static) methods.

Parameters

Name	Type	Description
inSubfolder	String	A path (relative to the app’s Documents directory) in which the app’s offline files will be stored. E.
formName	String	The name of the offline form (including “.htm” extension)
scafName	String	The App SCAF name (usually lower-cased version of the library name)
omnisWebUrl	String	The URL to the Omnis server, or web server. (If using a web server, you must provide omnisServer
omnisServer	String	If provided, the route to the Omnis server from the web server plugin.
omnisPlugin	String	If provided, the relative URL to the web server plugin, from the root of the web server.

Returns

None.

Example

```
omnis omnisInterface.scafHandler.initScafController(   inSubfolder: "offline",   formName: "jsOffline.htm",
scafName: "myapp",   omnisWebUrl: "https://mysite.com",   omnisServer: "192.168.1.123:9816",   omnisPlugin:
"/omnis_apache")
```

Example

```
omnis omnisInterface.scafHandler.initScafController(   inSubfolder = "offline",   formName = "jsOffline.htm",
scafName = "myapp",   omnisWebUrl = "https://mysite.com",   omnisServer = "192.168.1.123:9816",   omnisPlugin
= "/omnis_apache")
```

updateScafs()

updateScafs()

Updates SCAFs from the Omnis server configured in initScafController().
OMScafHandlerDelegate methods will be called with results.

Parameters

None.

Returns

None. (OMScafHandlerDelegate methods will be called with results)

loadOfflineForm()

loadOfflineForm([queryParams])

Load the configured offline form.

Parameters

Name	Type	Description
queryParams	Map<String, String>	A Map of key-value pairs to send as URL query parameters. These can be obtained the JavaScript: command to call: **jOmnis.getURLParameters()**This will return a JS & values.

or

Name	Type	Description
queryParams	[String: String]	A dictionary of key-value pairs to send as URL query parameters. These can be obt using the JavaScript: command to call: **jOmnis.getURLParameters()** This will retu these keys & values.

Returns

None.

OMScafHandlerDelegate

Provides callbacks regarding SCAF update progress etc.

Methods

Method Name	Description
onScafHandlerError(errorText, action)	(Optional) Called when an error occurs during the SCAF update/extraction process.
onScafUpdateCompleted(didUpdate, withErrors, newHtmlPath)	(Optional) Called when the SCAF update process completes (possibly with errors).
onOfflineModeConfigured(htmlPath)	Called when offline mode is first ready (either around startup time if offline files are already present, or after a successful SCAF update).

Enumerations

ScafAction

An enum containing the various actions the ScafHandler may execute.
Used in specifying which action failed when onScafHandlerError is called.

Key	Description
Unknown	No specific action.
CheckFiles	Checking for local HTML files or SCAF.
ExtractSCAF	Extracting a SCAF.
ReadSCAF	Reading the contents of a SCAF.
AccessForm	Accessing the offline form.
UpdateSCAFs	Updating SCAFs from the Omnis server.

onScafHandlerError()

onScafHandlerError(errorText, action)

(Optional) Called when an error occurs during the SCAF update/extraction process.

Parameters

Name	Type	Description
errorText	String	A description of the error which has occurred.
action	ScafAction	The action which failed.

Returns

Type	Description
Bool	True if you have handled the error, false if you wish default handling to occur. Default handling varies based on the action, but will generally show an OK message, or print a message to the console.

onScafUpdateCompleted()

onScafUpdateCompleted(didUpdate, withErrors, newHtmlPath)

(Optional) Called when the SCAF update process completes (possibly with errors).

Parameters

Name	Type	Description
didUpdate	Bool	Whether an update occurred. If this is false, and withErrors is empty, then no updates were a
withErrors	[String]	An array of error strings for any errors which occurred.

Returns

Type	Description
Bool	True if you have handled the message, false for default behaviour to occur. Default behaviour is to show an OK message with the results (Success, failure with errors, or no updates available).

Example

```
func onScafUpdateCompleted(didUpdate: Bool!, withErrors: [string]?, newHtmlPath: String!) -> Bool
{
  if (withErrors.count == 0)
  {
    let message = didUpdate ? "Update successful" : "No updates available"
    OMDialogs.showOKMessage(message: message, title: "SCAF Update")
  }
  else {
    OMDialogs.showOKMessage(message: withErrors.componentsJoined(by: ", "), title: "SCAF Update Error")
  }
  self.offlineFormPath = newHtmlPath
}
```

onOfflineModeConfigured()

onOfflineModeConfigured(htmlPath)

Called when offline mode is first ready (either around startup time if offline files are already present, or after a successful SCAF update).

Parameters

Name	Type	Description
htmlPath	String	The path to the directory containing the offline form. Each time a SCAF update occurs, this will change caching issues). Use the updated value passed to onScafUpdateCompleted to update any references y to this.

Returns

None.

Example

```
func onOfflineModeConfigured(htmlPath: String!) {
  self.offlineDir = htmlPath
  self.omnisInterface.scafHandler.loadOfflineForm()
}
```

OMDialogs

A class providing helper methods to display dialogs and loading overlays.

Everything is accessed statically.

Static Properties

Name	Type	Description
LOADING_OVERLAY_FONT	UIFont	The font used for text on loading overlays. Defaults to UIFont.sys
LOADING_OVERLAY_TEXT_SIZE	CGFloat	The text size used for loading overlays. Defaults to 16.
LOADING_OVERLAY_TEXT_COLOR	UIColor	The color used for text on loading overlays. Defaults to white.

Static Methods

Method Name	Description
showOKMessage(message, title)showOKMessage(context, message, title, callback)	Show a dialog with a single 'OK' button.
showDialog(message, title, buttons) showDialog(context, message, title, buttons)	Show a dialog with a list of defined buttons.
showYesNoMessage(message, title, onYes, onNo)showYesNoMessage(context, message, title, onYes, onNo)	Show a dialog with 'Yes' and 'No' buttons, with a handler for each.
showDialogForOmnisInterface(omnisInterface, data)	A helper function to show a dialog using the JSON data provided by the JS client for such dialogs.
showLoadingOverlay(onView, text, cancelAppearsAfter, cancelHandler)showLoadingOverlay(context, onViewID, inLayout, text, cancelAppearsAfter, cancelHandler)	Adds a dark loading overlay to a view. Blurs the background, and shows a centered Activity Indicator and optional text.
removeLoadingOverlay(onView) removeLoadingOverlay(onViewID, inLayout)	Removes any loading overlay from the passed view.

showLoadingOverlay()

showLoadingOverlay(context, onViewID, inLayout, [text, cancelAppearsAfter, cancelHandler])

Adds a loading overlay to a view.
Blurs the background, and shows a centered Activity Indicator with optional text and cancel handler.

Parameters

Name	Type	Description
context	Context	The Context in which to execute.
onViewID	Int	The Resource ID of the view to which the overlay should be added.
inLayout	ConstraintLayout	The layout which contains the view which should be overlaid. (Currently only supported)
text	String?	A message to display on the overlay.
cancelAppearsAfter	Long	Number of milliseconds to wait before showing a cancel button. Set to less t button to appear.
cancelHandler	() → Boolean	Callback function to call when the cancel button is pressed. Return **false** fro overlay to be removed automatically.

Returns

None.

Styling

The **colour** of the overlay background and text can be overridden by adding **color xml resources** named "omnis_loading_overlay_background" and "**omnis_loading_overlay_text"** to your project.

Similarly, the **text size** can be altered by adding a **dimen xml resource** named "**omnis_loading_overlay_text_size**".

showLoadingOverlay()

```
showLoadingOverlay(onView, [text, cancelAppearsAfter, cancelHandler])
```

Adds a loading overlay to a view.
Blurs the background, and shows a centered Activity Indicator with optional text and cancel handler.

Parameters

Name	Type	Description
onView	UIView	The view to which the overlay should be added.
text	String?	A message to display on the overlay.
cancelAppearsAfter	Long	Number of milliseconds to wait before showing a cancel button.Set to less than appear.
cancelHandler	() → Bool	Callback function to call when the cancel button is pressed. Return **false** from th to be removed automatically.

Returns

None.

Styling

The appearance of the loading overlay can be edited by changing the following static properties of OMDialogs:

Name	Type	Description
LOADING_OVERLAY_FONT	UIFont	The font used for text on loading overlays. Defaults to UIFont.sys
LOADING_OVERLAY_TEXT_SIZE	CGFloat	The text size used for loading overlays. Defaults to 16.
LOADING_OVERLAY_TEXT_COLOR	UIColor	The color used for text on loading overlays. Defaults to white.