ReportPlus Embedded
iOS SDK Guide

Disclaimer

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT ANY EXPRESS REPRESENTATIONS OF WARRANTIES. IN ADDITION, INFRAGISTCS, INC. DISCLAIMS ALL IMPLIED REPRESENTATIONS AND WARRANTIES, INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTURAL PROPERTY RIGHTS.

ReportPlus™ iOS v5.0 – Embedding Guide 1.0.

Table of Contents

Introduction

Welcome to the ReportPlus iOS Embedding Guide.

This document describes how to embed the ReportPlus v5 component into your iOS application. This component allows you to render dashboards created in the standalone ReportPlus application (either Desktop or iOS) by embedding the viewer component in your own iOS application.

Chapter 1
Planning your Implementation

Embedding Requirements

To embed the ReportPlus viewer component, you will need to meet the following requirements:

Xcode

You will need to have Xcode 7.3 or newer.

iOS version

ReportPlus requires iOS 8 or later. If your application supports previous versions, you will need to disable the ReportPlus component for those versions.

ReportPlus Frameworks

You will need to get a copy of the ReportPlus frameworks to include them in your application. The specific files you will need are:

- IG.framework
- IGChart.framework
- IGMF.framework
- SPFoundation.framework
- RPEngine.framework
- RPUI.framework
- RPConnectors.framework

ReportPlus SDK Sample

It is strongly advised to take a look at the RPSample application project. This is a very simple application that embeds ReportPlus to visualize dashboards.

Note: To gain access to the ReportPlus frameworks and SDK sample, please contact em-sales@infragistics.com.

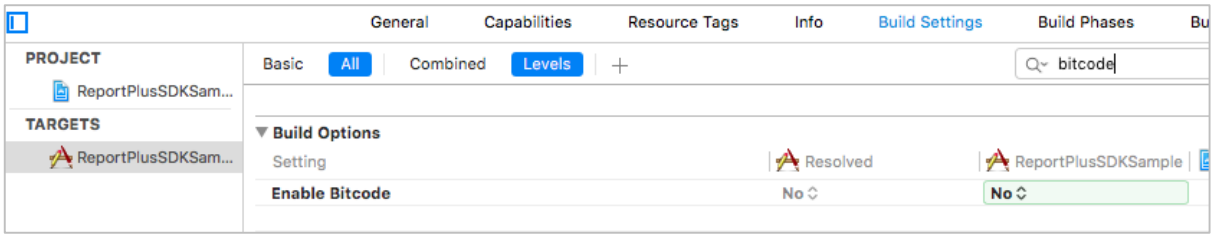
Preparing your Project

Modifying your Build Settings

Before adding the ReportPlus framework files to the Project embedded binaries, you will need to modify a setting in your Build Settings.

Steps

1. In **Xcode**, go to *Build Settings* for your application target.
2. Search for “bitcode”.
3. Set the “Enable Bitcode” setting to “No”.

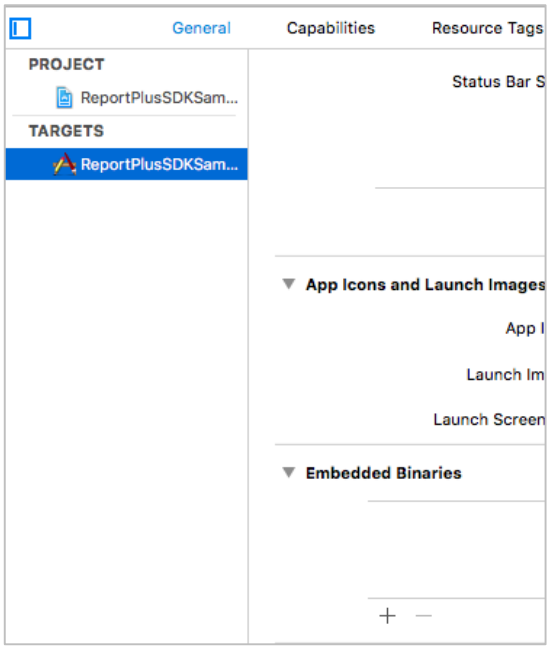


Adding the ReportPlus framework to your application

After modifying your Build Settings, you will be ready to add the ReportPlus frameworks to your application.

Steps

1. In **Xcode**, go to the application *Target Settings*.
2. Go to *General*.
3. Look for the “Embedded Binaries” option.

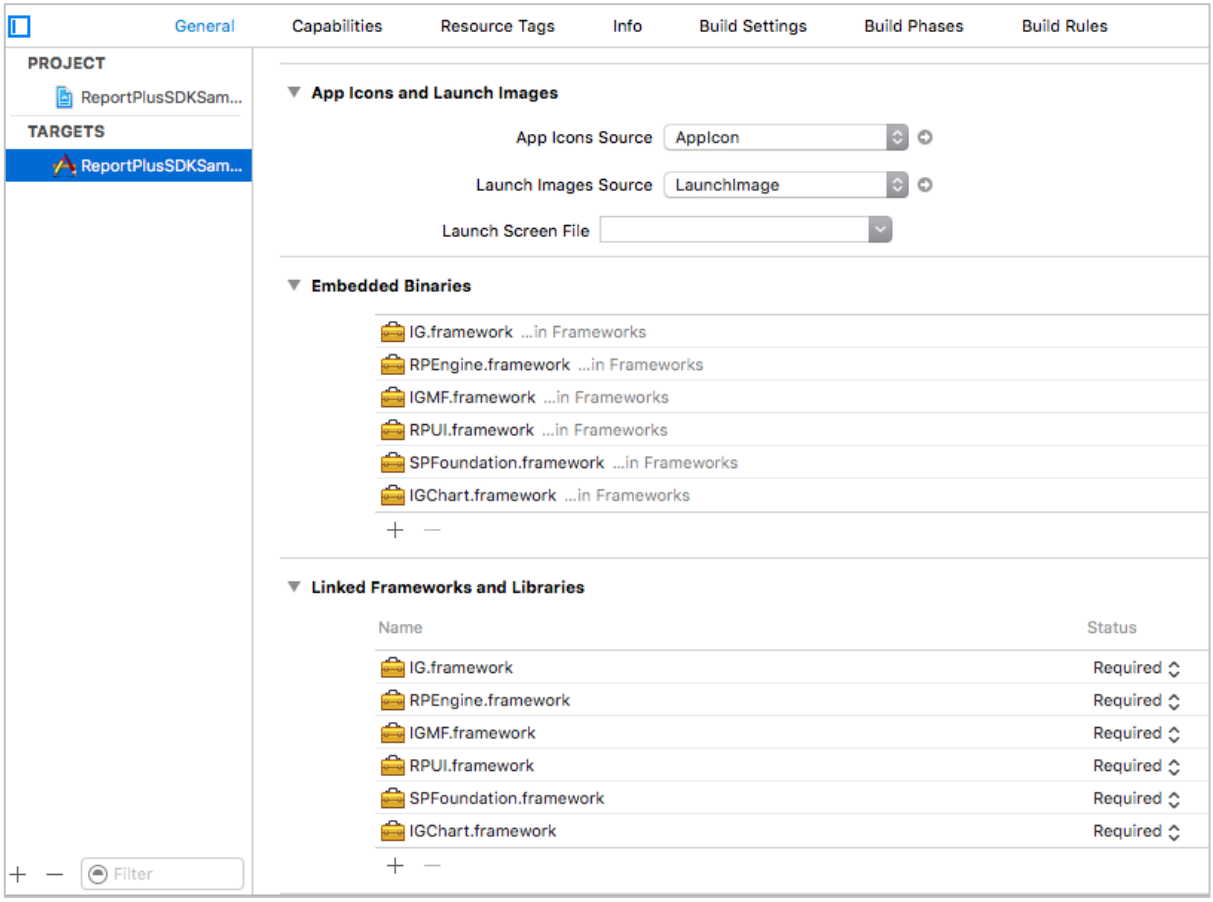


4. Press the “+” button and select all of the “.framework” files in the distribution:
    - IG.framework

- IGChart.framework
- IGMF.framework
- SPFoundation.framework
- RPEngine.framework
- RPUI.framework
- RPConnectors.framework (optional)

5. When prompted select *Create Groups* and move the references to the desired group in your project (like “Frameworks” or “Dependencies”).

The *General* tab should now look like the following:



As you can see, the frameworks are automatically added to the “Linked Frameworks and Libraries”. This means you are read to start using the ReportPlus component in your application.

> **Note:** RPEgnine framework includes connectors for SharePoint, CSV and Excel files, while RPConnectors.framework includes the rest of the connectors. That is why RPConnectors if optional; if you will use CSV or Excel files, stored locally or in a SharePoint server, you will not need it.

Adding the License File

To use the ReportPlus component in your application, you will need a license file (license.rplic), which you should have received along with the framework. The **license file is associated to the bundle for of your application**, so you will need a new license file if you change the bundle ID in the future.

The license file is read as a resource file, so you will need to add it as a resource in the *Build Phases section*.

Steps

1. In **Xcode**, go to *Build Phases* for your target.
2. Expand *Copy Bundle Resources*.
3. Press the "+" button.
4. Select the **license.rplic** file.

If you added the file before to your project, it should be available in the first screen. If not, you may need to press the "Add Other" button and browse to the location of the file in the file system.

Chapter 2
Using the ReportPlus Component

Using the ReportPlus Component

Now that your application includes the ReportPlus component, we will show you how to use it in your code.
You will need to include the "RPUI" framework; the other frameworks are used automatically by RPUI, so you only need to embed them to make them available.

Importing the Framework

Start by importing the framework in your code with the following command:

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

```
@import RPUI;
```

Using RPDashboardMainViewController

The main view controller class is RPDashboardMainViewController. Here is a code snippet that shows you how to use it:

```
NSData *data = //load the dashboard here;
RPDashboardMainViewController *vc =
     [[RPDashboardMainViewController alloc]
          initWithMode:RPDashboardMainViewControllerModeEmbedded];

//add the view controller and view to your app
[self addChildViewController:vc];
vc.view.frame = self.view.bounds;
[self.view addSubview:vc.view];

[vc prepareDashboardWithData:data
      onSuccess:^(RPPrepareDashboardResult *result) {
            [vc openPreparedDashboard];
      }
      onError:^(NSError *e) {
            NSLog(@"ERROR: %@", e);
      }
];
```

As depicted above, the opening is a two-steps process:

a. First, call a "prepare" method that will run asynchronously.
b. When this method returns, you can call the "openPreparedDashboard" method to display the dashboard. You may want to display some kind of activity indicator while this process runs.
c.

The ReportPlus component only handles the visualization of a dashboard, not the storage, so it's up to your application to read dashboard files from the place where they are stored. This is the reason why, in your first line, **you must fill that in order to read a dashboard file.**

Sample Application

You can take a look at the sample application to review this functionality. A few sample dashboards are embedded into the application and get loaded from the main bundle.

Passing parameters to a dashboard

A dashboard defines **input parameters**, which are automatically defined for each Global Filter defined in the dashboard.

If you take a look to the "HR" sample included in the distribution, it contains two Global Filters: "Office" and "Departments". This means that when you open the "HR" dashboard you can (optionally) specify two parameters:

```
vc.dashboardParameters = @{@"Office": @"Tokyo, Japan", @"Departments": @"IxD"};
```

You may specify a single parameter, there is no need to specify both:

```
vc.dashboardParameters = @{@"Office":@"Tokyo, Japan"};
```

In the previous examples, we set the same field used as the display field in the Global Filter for each parameter (like the name of the Office). However, you may want to pass another field, like the ID of the selected Office:

```
vc.dashboardParameters = @{@"Office.OfficeId": @"5"};
```

Just take into account that "OfficeId" must be a field of the dataset dropped to the Global Filter when the dashboard was created.

Locating the local files referenced by the dashboard

Your dashboards will probably use some local resources (like a local Excel spreadsheet or a CSV file). The ReportPlus component looks for these files in the "Documents" folder of your application.

You can get that location with this code:

```
[NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                     NSUserDomainMask, YES) objectAtIndex:0];
```

Additionally, you can change that location by calling the following method:

```
/**
 * Change the location for the local files provider, this directory must exist in
advance
 */
+ (void)setLocalDocumentsBaseDir:(NSString *)path;
```

You need to put (or update) your files in that location before opening the dashboard.
Dashboards may also embed these files, a dashboard file (with .rplus extension) is actually a zip file, and will optionally include data files in the “Attached Files” folder.
By default these files are automatically imported to the Documents folder but you can tweak how it works with these properties:

```
/**
 * Whether to import the data files included in the dashboards or not. Defaults to
YES
 */
@property (nonatomic, assign) BOOL importAttachedFiles;
/**
 * Whether to ask before overwriting an embedded file. If NO the file will be
overwritten without prompt. Defaults to NO.
 */
@property (nonatomic, assign) BOOL askForAttachedFilesOverwrite;
```

Connecting to External Data Sources

As stated before, RPEngine.framework includes the connectors to open CSV and Excel files locally or from a SharePoint site, while RPConnectors.frameworks provides support for the rest of the connectors supported by ReportPlus (MS SQL Server, MySQL, MS Analysis Services and others).

ReportPlus is a mobile component that connects directly to data sources, which means that if, for instance, you are opening a dashboard that loads data from a MySQL database, your application needs to be able to establish a direct connection to the database server. This is usually achieved by running the app in the same network where the database server resides or by using a VPN.

In order to connect to these data sources ReportPlus needs to know which credentials to use for authentication when connecting to the server.

There are two ways to do this:

1. Asking the user for credentials
2. Providing credentials programmatically

Asking the User for Credentials

In the "Prepare" phase, ReportPlus looks for all the data sources used by a dashboard when opening it, and determines if there are credentials already specified for them. The list of data sources without associated credentials is returned back as part of the "RPPrepareDashboardResult" object.

You can use the following code to display the standard "Import" dialog, which will ask the user to enter their credentials for each data source with no associated credentials:

```
[vc prepareDashboardWithData:data
                   onSuccess:^(RPPrepareDashboardResult *result) {
    if (result.missingDataSources.count > 0) {
        UIViewController *importVc =
         [vc importViewControllerWithPrepareResult:result
            onCompletion:^(UIViewController *ivc) {
                 [ivc dismissViewControllerAnimated:YES completion:^{
                     [vc openPreparedDashboard];
                 }];
            } onCancel:^(UIViewController *ivc) {
                 [ivc dismissViewControllerAnimated:YES completion:nil];
            }];
        UINavigationController *nav = [[UINavigationController alloc]
            initWithRootViewController:importVc];
        nav.modalPresentationStyle = UIModalPresentationFormSheet;
         [self presentViewController:nav animated:YES completion:nil];
    } else {
         [vc openPreparedDashboard];
    }
} onError:^(NSError *e) {
    NSLog(@"ERROR: %@", e);
}];
```

As you can see in the code above, you display the Import view controller if needed. When it completes, you dismiss it and open the dashboard.

Providing credentials programmatically

The previous example shows the easiest way to ask for credentials because you are reusing the existing code in the ReportPlus component. However, you may want to customize this experience in order to:

- Ask the user for credentials using your own UI.
- Provide credentials the user already specified to access your application.
- Provide credentials that the user does not know about, like hardcoded credentials in your application.

You can achieve this by configuring an "authentication provider" in the main view controller.

```
    vc.authenticationProvider = self;
```

This is the only method to implement as part of the "RPDashboardAuthenticationProvider" protocol:

```
- (void)credentialsForDataSource:(RPDashboardDataSource *)ds
                       onSuccess:(RPCredentialsCallback)callback;
```

This method will be called for each associated data source without credentials. You must call the callback block in order to the "Prepare" process to continue, but you can call it passing `nil` as the parameter if you don't have credentials or if the user cancelled the input process.

You can use the following properties in the RPDashboardDataSource class to identify the data source for which the credentials are being requested:

```
@interface RPDashboardDataSource : NSObject
    @property (nonatomic, retain) NSString *dataSourceId;
    @property (nonatomic, retain) NSString *title;
    @property (nonatomic, retain) NSString *provider;
    @property (nonatomic, retain) NSDictionary<NSString *,
                                               NSObject *> *properties;
@end
```

- "title" is the name entered by the user when creating the data source.
- "dataSourceId" is a GUID automatically generated when the data source was created and that uniquely identifies it.

> **Note:** At the end of this document, you will find a table with the list of provider names and the properties supported for each of them.

The following code shows how to ask for credentials using an UIAlertController with three text fields:

```
- (void)askForCredentials:(RPDashboardDataSource *)ds
             onCompletion:(RPCredentialsCallback)callback {

     NSString *host = (NSString *)ds.properties[@"Host"];
     UIAlertController *alertController = [UIAlertController

        alertControllerWithTitle:@"Authentication"
        message:[NSString stringWithFormat:@"Enter your credentials for SQL
Server at %@", host]
        preferredStyle:UIAlertControllerStyleAlert];

     [alertController addTextFieldWithConfigurationHandler:^(UITextField
*textField) {
       textField.placeholder = @"Domain";
     }];
     [alertController addTextFieldWithConfigurationHandler:^(UITextField
*textField) {
       textField.placeholder = @"User";
     }];
     [alertController addTextFieldWithConfigurationHandler:^(UITextField
*textField) {
       textField.placeholder = @"Password";
       textField.secureTextEntry = YES;
     }];
     UIAlertAction *okAction = [UIAlertAction actionWithTitle:@"Login"
style:UIAlertActionStyleDefault handler:^(UIAlertAction *action) {
       UITextField *domain = alertController.textFields[0];
       UITextField *user = alertController.textFields[1];
       UITextField *password = alertController.textFields[2];
       if (user.text.length > 0 && password.text.length > 0) {
             callback([[RPDashboardCredentials alloc] initWithDomain:domain.text
user:user.text password:password.text]);
       } else {
             callback(nil);
       }
     }];
     UIAlertAction *cancelAction = [UIAlertAction actionWithTitle:@"Cancel"
style:UIAlertActionStyleCancel handler:^(UIAlertAction *action) {
       callback(nil);
     }];
     [alertController addAction:cancelAction];
     [alertController addAction:okAction];

     [self presentViewController:alertController animated:YES completion:nil];
}
```

Modifying Data Sources before opening the dashboard

Suppose you are using the WebResource data provider to retrieve a CSV file from the server. To do this, you connect to a server component that returns the data to be used by the dashboard.

For instance, you might want to modify the URL to be used to get the data from the server to pass information about the user who is currently logged in. You have different ways to achieve this:

- **Creating the dashboard referencing a CSV file locally instead of using a Web Resource**. In this case, your application either downloads or generates the CSV locally based on the current user before opening the dashboard.
- **Customizing the credentials for the Web Resource data source** as described before. On the server side, you return the data tailored to the authenticated user.
- **Modifying the URL used by the Web Resource** data source to append the username parameter.

Modifying the URL used by the Web Resource

To do this, you can specify a data source provider to the main view controller:

```
    vc.dataSourceProvider = self;
```

This is the only method to implement as part of the "RPDashboardDataSourceProvider" protocol:

```
- (void)replacementForDataSource:(RPDashboardDataSource *)ds
                       onSuccess:(RPDataSourceCallback)callback;
```

This procedure is very similar to the authentication provider one. Once you implement this method, you must call the callback block. If you don't want to modify this data source, just call the callback block with `nil` as the parameter.

The following code modifies all of the SQL Server data sources to change the host to a fixed IP address, and all of the Web Resource data sources to append the login for the current user to the URL.

```
- (void)replacementForDataSource:(RPDashboardDataSource *)ds
                      onSuccess:(RPDataSourceCallback)callback {
     if ([ds.provider isEqualToString:@"SQLSERVER"]) {
        NSMutableDictionary *newProperties = [[NSMutableDictionary alloc]
initWithDictionary:ds.properties];
        newProperties[@"Host"] = @"10.0.0.20";
        ds.properties = newProperties;
        callback(ds);
     } else if ([ds.provider isEqualToString:@"WEBSERVICE"]) {
        NSString *url = (NSString *)ds.properties[@"Url"];
        NSString *login = @"";//TODO: use the current user here
        NSMutableDictionary *newProperties = [[NSMutableDictionary alloc]
initWithDictionary:ds.properties];
        newProperties[@"Url"] = [NSString stringWithFormat:@"%@?user=%@", url,
login];
        ds.properties = newProperties;
        callback(ds);
    } else {
        callback(nil);
    }
}
```

Appendices

Appendix 1: Supported Properties on Data Sources for each Provider
Appendix 2: Document Changelog

Appendix 1: Supported Properties on Data Sources for each Provider

All properties are required unless specified otherwise.

Provider	Provider Name	Supported Properties
AppFigures	APPFIGURES	No properties

MS Analysis Services	ANALYSISSERVICES	• Mode: string "HTTP" or "Native"
		• For HTTP
		  ○ Url: string
		• For Native:
		  ○ Host: string
		Port: int (optional, defaults to 2383)

Dropbox	DROPBOXPROVIDER	DropboxId: string
MS Dynamics CRM	DYNAMICS_CRM	Url: string
Google Drive	GOOGLEDRIVEPROVIDER	No properties
Google Analytics	GOOGLE_ANALYTICS	No properties
Facebook	FACEBOOK	No properties
Flurry	FLURRY	ApiAccessCode: string
IMAP	IMAP	• Host: string • Port: int (optional, defaults to 993 for SSL and 143 for non-SSL) Security: string "None", "SSL", "StartTLS"
MySQL	MYSQL	• Host: string Port: int (optional, defaults to 3306)
OData	ODATAPROVIDER	Url: string
Oracle	ORACLE	• Host: string • Port: int (optional, defaults to 1521) • SID: string • SERVICE_NAME:string One of SID or SERVICE_NAME must be specified
PostgreSQL	POSTGRES	• Host: string • Port: int (optional, defaults to 5432) Database: string (optional)
Salesforce	SALESFORCE	Organization: string
Sharepoint	SHAREPOINT	• Url: string AuthenticationMethod: string (optional, one of "Windows", "Form", "Office365", "WebLogin" and defaults to "Windows")
MS SQL Server	SQLSERVER	• Host: string • Port: int (optional, defaults to 1433) Database: string (optional)
MS Reporting Services	SSRS	• Url: string • Path: string (optional)

		• ServerMode: string (optional, one of “Native”, “Integrated”) ServerVersion: string (optional, one of “SSRS_SERVER_2005”, “SSRS_SERVER_2008R2”). Determined automatically
Sybase	SYBASE	• Host: string Port: int (optional, defaults to 5000)
Twitter	TWITTER	No properties
UserVoice	USERVOICE	Url: string
Web Resource	WEBSERVICE	Url: string

Appendix 2: Document Changelog

Version	Chapter	Section	Description
1.0	All chapters	All sections	Document creation.