

iOS Blocks

Application Programming Interface

Contents:

1. Overview
2. Design considerations
3. Content view
4. Buttons view
5. Alternative icon view
6. Additional optional methods
7. Widget directory layout
8. Info.plist options

Overview

An iOS Blocks widget is simply a class that implements the methods given by the provided delegate. In the provided templates for both theos and iOSOpenDev, this class is a subclass of UIViewController.

There are a number of both required and optional delegate methods to implement in your custom widget:

```
@protocol IBKWidget <NSObject>
```

```
@required
```

```
-(UIView*)viewWithFrame:(CGRect)frame isIpad:(BOOL)isIpad;  
-(BOOL)hasButtonAreaView;  
-(BOOL)hasAlternativeIconView;
```

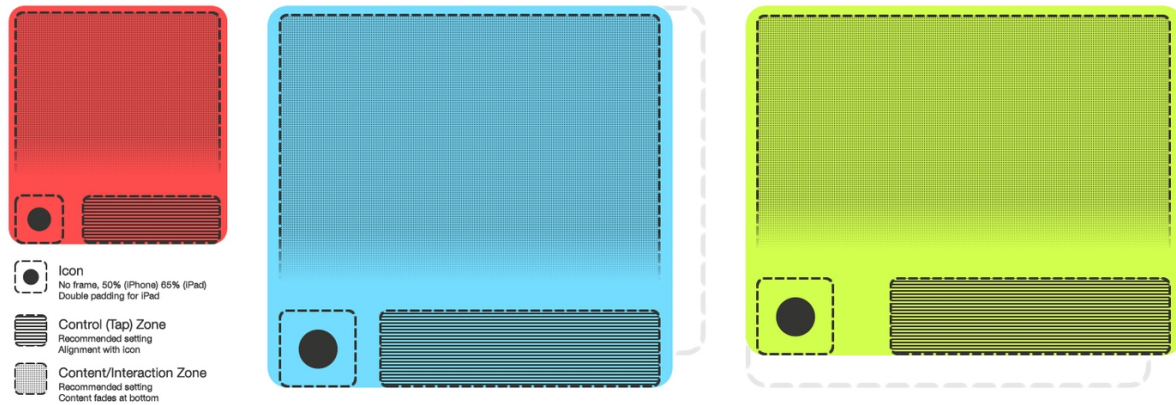
```
@optional
```

```
-(UIView*)buttonAreaViewWithFrame:(CGRect)frame;  
-(BOOL)wantsNoContentViewFadeWithButtons;  
-(UIView*)alternativeIconViewWithFrame:(CGRect)frame;  
-(NSString*)customHexColor;  
-(void)willRotateToInterfaceOrientation:(int)orient;  
-(void)didRotateToInterfaceOrientation:(int)orient;
```

In addition to these, there are options that can be specified within the Info.plist file of your widget's bundle.

Design considerations

When designing and building your widget, you need to adhere to a few design criteria set out by the original designer, @technofou:

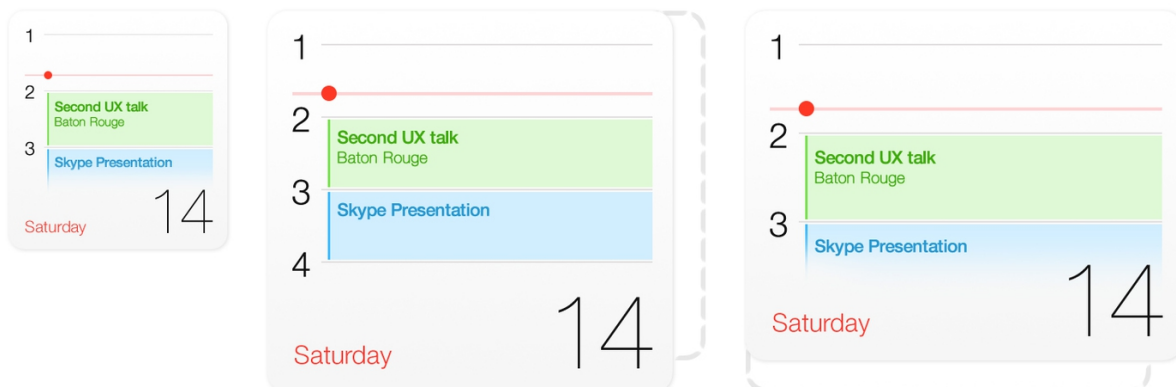


Each widget should be interacted with by gestures, and any tap gesture should only be registered in the view returned by:

```
-(UIView*)buttonAreaViewWithFrame:(CGRect) frame;
```

In addition, launching of an app is handled only by the user tapping on the icon area in the lower left; this is handled by the iOS Blocks framework. Also, if a button view is made available by your widget, then the content area will be faded out at the bottom as above.

Of course, this doesn't need to be adhered to exactly; some widgets may have functions that necessitate tapping on the content view for example. Additionally, don't feel fully constrained to the exact layout of buttons, icon and content – the in-built Calendar widget doesn't, as shown below:



Content View

The content view is the view returned by:

```
-(UIView*)viewWithFrame:(CGRect)frame isIpad:(BOOL)isIpad;
```

This method is called with the frame variable being the size of the widget, and is also called first within the framework, hence providing it with the `isIpad` variable.

When building your widget, you should always assume that the height of this view will be constrained to just above the icon's height, as given by:

```
[objc_getClass("IBKAPI") heightForContentView];
```

The need for the view to be the size of the widget is if a custom background is applied to the content view, and this is wanted to cover the entire widget. The constraint however is so that content such as table view cells are not covered by the icon view which may look odd, and so that content isn't displayed underneath the faded area when buttons are in use. That said, using the constraint is optional.

When on a device that can have a rotating SpringBoard, the size of the content view's frame will be modified. As a result, it is recommended to use a subclass of `UIView` for your content view, as then you can override

```
-(void)layoutSubviews;
```

to correctly re-layout the subviews with your view's new size.

Buttons View

The buttons view is returned by:

```
-(UIView*)buttonAreaViewWithFrame:(CGRect) frame;
```

To specify that your widget requires this view, you must also return **YES** in

```
-(BOOL)hasButtonAreaView;
```

This view will never have its frame change size during rotation, so you will not need to modify the positioning and size of any element.

The height of the button view will always be the same as that of the icon view/image.

Additionally, the iOS Blocks framework will automatically apply a fading mask to the bottom of the content view if

```
-(BOOL)hasButtonAreaView;
```

returns **YES**. If you wish to disable that, you will need to return **YES** in

```
-(BOOL)wantsNoContentViewFadeWithButtons;
```

Alternative Icon View

The alternative icon view allows you to specify a custom view for your widget's icon. This can be anything; the in-built Calendar widget for example displays the current day here.

To specify that your widget will return a custom icon view, you will need to return YES in

```
-(BOOL)hasAlternativeIconView;
```

and then also implement

```
-(UIView*)alternativeIconViewWithFrame:(CGRect)frame;
```

As with the buttons view, this will not change frame size during rotation, and will also keep the same origin too. Additionally, this view does not mask to bounds, so you may extend beyond the given bounds. Bear in mind though that the touch area for launching the icon will not extend beyond the frame given.

Additional optional methods

In addition to the methods already covered by this document, there are also other methods that you can implement in your widget that will affect how the framework will treat it.

For now, this is only the below method:

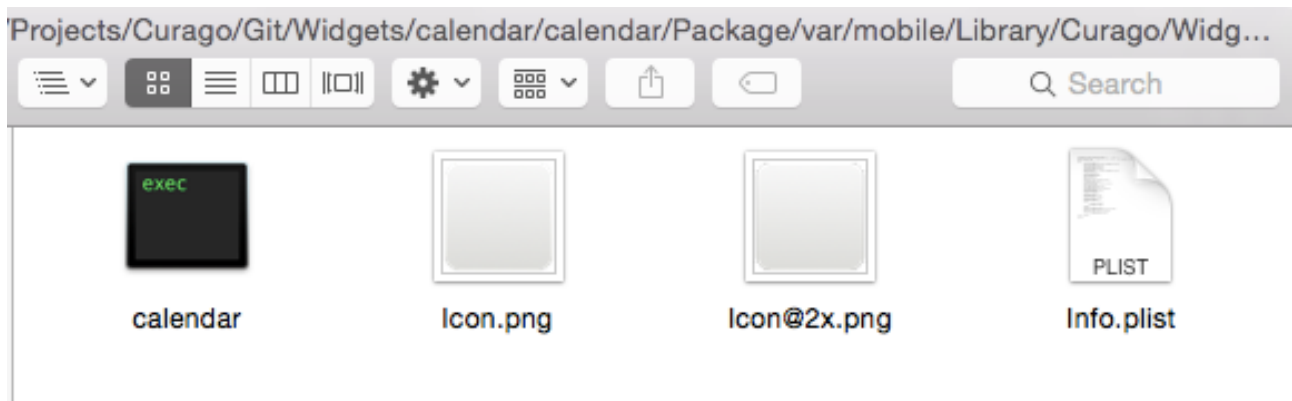
```
-(NSString*)customHexColor;
```

Implementing this will allow you to specify a custom colour in hex for the background of the widget. Nothing fancy, just do:

```
return @"FFFFFF";
```

Widget directory layout

Each widget will have an Info.plist file and an executable at a minimum, like so:



Additionally, icon files are not mandatory; if none are present, then the app's icon will be used instead. The iOS Blocks framework will automatically search for an Icon.png with the appropriate suffix for device resolution.

The folder name for your widget should be like so:

`<your_dev_name>.<app_bundle_id>`

e.g.

`matchstic.com.clickgamer.AngryBirds`

The reasoning for this is so that multiple widgets for the same app may be installed, and then the user can choose which to use in the iOS Blocks settings.

Default widgets provided by iOS Blocks will omit the `<your_dev_name>` prefix.

All widgets are located in `/var/mobile/Library/Curago/Widgets`

Info.plist Options

Within each widget's directory is an Info.plist which also contains a few additional options; most options are only relevant when using HTML instead of Objective-C, which is covered by another document. Only those relevant are given here.

CustomGameWidget : boolean

Specify true if you are making a widget for a game, else iOS Blocks will display it's own game widget rather than yours.

CustomColor : string

A hex string representing the background colour of the widget. This is not necessary if you implement the given protocol method for this.

WantsNotificationsTable : boolean

Specify true if you wish to use iOS Blocks' in-built notifications widget, but also want to customise it with a custom background colour or icon image. Bear in mind that you cannot load up any code if this is true.