

[ClockKit](#) /  / [Graphic](#) / Displaying essential information on a watch face

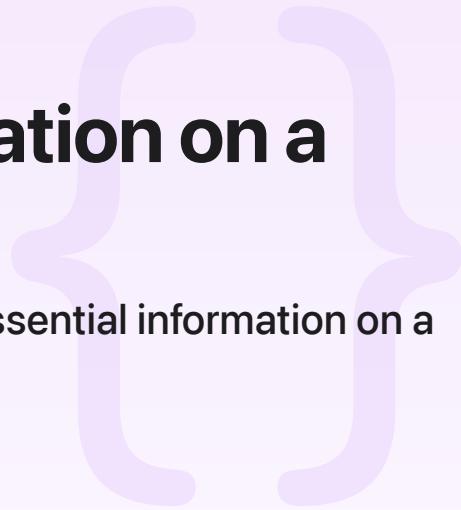
Sample Code

Displaying essential information on a watch face

Implement complications in a watch app to display essential information on a watch face.

[Download](#)

iOS 13.0+ | iPadOS 13.0+ | watchOS 6.1+ | Xcode 14.2+



Overview

This sample code project shows how to present time-based information on the Infograph or Infograph Modular watch face using Graphic Corner, Graphic Circular, Graphic Bezel, and Graphic Rectangular complication families.

Complications are small elements that display essential information (like weather, calendar, and activity tracking) right on the watch face. Each complication needs a data source, or a timeline, that describes what content to present and when. In this sample, the timeline contains 24 events named E00, E01,, E23, each of which lasts one hour and occurs every day. Each complication presents the progress and other essential information of the currently occurring event in its own visual form.

A complication family can have multiple variants, and so this sample app provides a list for users to select and play with different variants. You can run the app on a watch that supports graphic complications, tap a family and pick a variant, then tap the Apply button to save the configuration and refresh the complications.

Configure the sample code project

To configure the sample code project, do the following in Xcode:

1. Select the Complications target, then change the bundle ID to <Your iOS app bundle ID>. Select the appropriate team to let Xcode automatically manage your provisioning profile. See [Assign the project to a team](#) for more details.
 2. Repeat step 1 for the WatchKit app and WatchKit Extension target. The bundle IDs should be <Your iOS app bundle ID>.watchkitapp and <Your iOS app bundle ID>.watchkitapp.watchkitextension respectively.
 3. Open the Info.plist file of the WatchKit Extension target, navigate to NSExtension > NSExtensionAttributes > WKAppBundleIdentifier key, and change the value of the key to <Your iOS app bundle ID>.watchkitapp.
 4. Select the Complications WatchKit App target and the device in Xcode's schema field, then build and run the target on your device.

To show a complication on the watch:

1. Select the Infograph or Infograph Modular watch face on the watch.
 2. Firmly press the watch face to show the customization screen, then tap the Customize button.
 3. Swipe left to show the complication configuration screen, then tap a complication.
 4. Rotate the Digital Crown to choose the Complications app.
 5. Press the Digital Crown and tap the screen to go back to the watch face.

See [Customize the watch face](#) in the Apple Watch User Guide to learn how to configure the watch face.

Provide timeline entries for the complications

To keep the complications up to date, this sample implements the following [CLKComplicationDataSource](#) methods to provide the supported timeline travel direction, the timeline end date, and the future timeline entries for all complications:

- `getSupportedTimeTravelDirections(for:withHandler:)` The sample implements this method and passes `.forward` to the handler to tell the system this app can provide future data entries for all its complications.

```
func getSupportedTimeTravelDirections(for complication: CLKComplication,  
                                      withHandler handler: @escaping (CLKComplication  
handler([.forward])  
{
```

- `getTimelineEndDate(for:withHandler:)` The sample implements this method to provide the end date of the prepared timeline. The system doesn't attempt to retrieve the future timeline entries if this method doesn't have an implementation.

```
func getTimelineEndDate(for complication: CLKComplication,
                       withHandler handler: @escaping (Date?) -> Void) {
    let delegate: ExtensionDelegate! = WKExtension.shared().delegate as? ExtensionDelegate
    let endTime = delegate.timeline.endDateTime(after: Date())
    handler(endTime)
}
```

- `getTimelineEntries(for:after:limit:withHandler:)` The sample implements this method to provide the future timeline entries. Each time the system calls this method, the sample provides the entries for the next day after the specified date time.

```
func getTimelineEntries(for complication: CLKComplication, after date: Date, limit: Int,
                       withHandler handler: @escaping ([CLKComplicationTimelineEntry]?) -> Void) {
    let entries = nextDayEntries(for: complication, after: date, limit: limit)
    handler(entries)
}
```

After getting the timeline entries from the app, the system caches them and progresses through them automatically. After progressing through a reasonable amount data, the system wakes up the watch app and calls `getTimelineEntries(for:after:limit:withHandler:)` again for more entries.

Create complication templates

Each timeline entry an app provides to the system has to contain a complication template, which is an instance of a `CLKComplicationTemplate` subclass describing the content and visual form of the complication. This sample provides a template for each variant of the graphic complication families. A real-world app may just support the variant that best fits the content it presents.

Creating a complication template involves creating a template object and setting up its properties. The following code creates a `CLKComplicationTemplateGraphicCornerStackText` for a graphic corner complication:

```
let template = CLKComplicationTemplateGraphicCornerStackText()
template.outerTextProvider = CLKSimpleTextProvider(text: event.name)
template.innerTextProvider = CLKRelativeDateTextProvider(date: event.startDateTime(), style: .natural,
```

```
units: [.hour, .minute, .second]
```

```
template.innerTextProvider.tintColor = .orange
```

The system supports multicolor text and full-color images, which can make a complication really vivid. Be aware that Infograph and Infograph Modular watch faces have multicolor and tinted modes that users can select when customizing the faces, and that the system automatically desaturates images on a tinted watch face by default. To avoid showing a desaturated image, this sample uses [CLKFullColorImageProvider](#) to provide both full-color and tinted versions.

```
let tintedImageProvider = CLKImageProvider(onePieceImage: tintForeground,  
                                            twoPieceImageBackground: tintBackground,  
                                            twoPieceImageForeground: tintForeground)  
let fullColorImageProvider = CLKFullColorImageProvider(fullColorImage: fullColorImage,  
                                                       tintedImageProvider: tintedImageProvider)
```

Multicolor text providers display a single color in the tinted mode. This sample uses the following code to create a piece of text for the `body1TextProvider` property of the [CLKComplicationTemplateGraphicRectangularStandardBody](#) template, which highlights the relative date text in orange in the multicolor mode, and displays the white color in the tinted mode.

```
let part1 = CLKSimpleTextProvider(text: "HAS ")  
part1.tintColor = .blue  
let part2 = CLKRelativeDateTextProvider(date: endDateTime, style: .naturalAbbreviated)  
part2.tintColor = .orange  
let part3 = CLKSimpleTextProvider(text: " TO GO")  
part3.tintColor = .green  
template.body1TextProvider = CLKTextProvider(format: "%@%@", part1, part2, part3)
```