**1. Temp**

* **WidgetKit**

To add a new widget, go to File -> New -> Target to create a target. To make a class or struct accessible from the target, select the file and add it to the Target Membership list.

Graphical user interface, text, application, email

Description automatically generated

To pass data from your app to the widget, go to Target -> Signing & Capabilities, add a new App group to both the app and the target with the same name. The following code saves data to the UserDefaults and reload the widget.

|  |
| --- |
| **@AppStorage**("item", **store**: **UserDefaults**(**suiteName**: "group.Cicimaya.WidgetExample"))  var itemData: Data = Data()  private func refreshWidget(item: Item) {  guard let data = try? JSONEncoder().encode(item) else {  return  }    itemData = data  **WidgetCenter.shared.reloadAllTimelines**()  } |

On the widget's side, read data from the UserDefaults in the provider.

|  |
| --- |
| struct Provider: TimelineProvider {    **@AppStorage**("item", **store**: **UserDefaults**(suiteName: "group.Cicimaya.WidgetExample"))  var itemData: Data = Data()    private func getItem() -> Item? {  return try? JSONDecoder().decode(Item.self, from: itemData)  }  } |

* **Widget**

Your widget should adopt the Widget protocol. There are two kinds of configurations.

* StaticConfiguration. It is used for a widget with no user-configurable properties.
* IntentConfiguration. It is used for a widget with user-configurable properties.

|  |
| --- |
| **@main**  struct MyWidget: **Widget** {  var body: some **WidgetConfiguration** {  **StaticConfiguration**(kind: "MyWidget", provider: **Provider**()) { entry in  MyWidgetEntryView(entry: entry)  }  .**configurationDisplayName**("My Widget")  .**description**("This is an example widget.")  .**supportedFamilies**([.systemSmall, .systemMedium, .systemLarge])  }  } |

The following shows how to support multiple widgets.

|  |
| --- |
| **@main**  struct MyWidgets: **WidgetBundle** {  **@WidgetBundleBuilder**  var body: some Widget {  MyWidget1()  MyWidget2()  MyWidget3()  }  } |

* **TimelineEntry**

A timeline entry specifies the date and data to display on the widget.

|  |
| --- |
| struct ItemEntry: **TimelineEntry** {  let date = Date()  let item: Item  } |

* **TimelineProvider**

A TimelineProvider creates timeline entries with dates that tell when to update the widget's content. If you want the widget to be user customizable, adopt the provider to the IntentTimelineProvider protocol.

|  |
| --- |
| struct Provider: **TimelineProvider** {  func **getTimeline**(in context: Context, completion: @escaping (Timeline<Entry>) -> Void) {  guard let item = getItem() else {  return  }    let entry = **ItemEntry**(item: item)  let timeline = **Timeline**(entries: [entry], policy: .**never**)  completion(timeline)  }  } |

The following generates a timeline consisting of five entries an hour apart.

|  |
| --- |
| func **getTimeline**(in context: Context, completion: @escaping (Timeline<Entry>) -> Void) {  var entries: [ItemEntry] = []  let currentDate = Date()  for i in 0 ..< 5 {  let date = Calendar.current.date(byAdding: .hour, value: i, to: currentDate)!  let entry = ItemEntry(date: date)  entries.append(entry)  }  let timeline = Timeline(entries: entries, policy: .**atEnd**)  completion(timeline)  } |

The following generates a timeline and requests a new timeline in 15 minutes.

|  |
| --- |
| func **getTimeline**(in context: Context, completion: @escaping (Timeline<Entry>) -> Void) {  let entry = ItemEntry(item: item)  let nextUpdateDate = Calendar.current.date(byAdding: .minute, value: 15, to: Date())!  let timeline = Timeline(entries: [entry], policy: .**after**(nextUpdateDate))  completion(timeline)  } |

The getSnapshot method provides a timeline entry that represents the current time and state of a widget.

|  |
| --- |
| func **getSnapshot**(in context: Context, completion: @escaping (ItemEntry) -> ()) {  guard let item = getItem() else {  return  }    completion(**ItemEntry**(item: item)  } |

The placeholder method provides a timeline entry representing a placeholder version of the widget.

|  |
| --- |
| func **placeholder**(in context: Context) -> ItemEntry {  **ItemEntry**(item: Item(name: "Rose", description: "I love you!"))  } |

* **Widget View**

A widget defines its content using a SwiftUI view.

|  |
| --- |
| struct **MyWidgetEntryView** : View {  **@Environment(\.widgetFamily)** var family  var entry: Provider.Entry  **@ViewBuilder**  var body: some View {  switch family {  case .**systemSmall**:  MyWidgetSmallView(entry: entry)  case .**systemMedium**:  MyWidgetMediumView(entry: entry)  default:  MyWidgetLargeView(entry: entry)  }  }  } |

The following displays a date as timer counting from now.

|  |
| --- |
| let date = Calendar.current.date(**byAdding**: .day, value: 1, to: **Date**())!  Text(date, **style**: .**timer**)  .font(.system(.title, design: .monospaced)) |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |